El gran libro de HTML5, CSS3 & Javascript

Juan Diego Gauchat

marcombo
ediciones técnicas

El gran libro de HTML5, CSS3 & Javascript

Juan Diego Gauchat

marcombo
ediciones técnicas

El gran libro de
HTML5, CSS3 y Javascript

J. D. Gauchat

El gran libro de HTML5, CSS3 y Javascript

Segunda edición, 2013

© 2013 Juan Diego Gauchat

© 2013 MARCOMBO, S.A.

www.marcombo.com

Los códigos fuente para este libro se encuentran disponibles en www.minkbooks.com.

«Cualquier forma de reproducción, distribución, comunicación pública o transformación de esta obra solo puede ser realizada con la autorización de sus titulares, salvo excepción prevista por la ley. Diríjase a CEDRO (Centro Español de Derechos Reprográficos, www.cedro.org) si necesita fotocopiar o escanear algún fragmento de esta obra».

ISBN: 978-84-267-2064-1

Dedicado a aquellos

que ya no están a mi lado.

Índice

Introducción

1. Documentos HTML5

1.1 Componentes básicos

1.2 Una breve introducción a HTML

1.2.1 Etiquetas y elementos

1.2.2 Atributos

1.2.3 Elementos anteriores

1.3 Estructura global

1.3.1 <!DOCTYPE>

1.3.2 <html>

1.3.3 <head>

1.3.4 <body>

1.3.5 <meta>

1.3.6 <title>

1.3.7 <link>

1.4 La estructura del cuerpo del documento

1.4.1 Organización

1.4.2 <header>

1.4.3 <nav>

1.4.4 <section>

1.4.5 <aside>

1.4.6 <footer>

1.5 En el interior del cuerpo

1.5.1 <article>

1.5.2 <hgroup>

1.5.3 <figure> y <figcaption>

1.5.4 <details> y <summary>

1.6 Elementos nuevos y elementos antiguos

1.6.1 <mark>

1.6.2 <small>

1.6.3 <cite>

1.6.4 <address>

1.6.5 <wbr>

1.6.6 <time>

1.6.7 <data>

1.7 Nuevos atributos y viejos atributos

1.7.1 El atributo data–*

1.7.2 reversed

1.7.3 ping y download

1.7.4 translate

1.7.5 contenteditable

1.7.6 Spellcheck

2. Estilos CSS y modelos de caja

2.1 CSS y HTML

2.2 Breve introducción a CSS

2.2.1 Reglas CSS

2.2.2 Propiedades

2.2.3 Estilos en línea

2.2.4 Estilos incrustados

2.2.5 Archivos externos

2.2.6 Referencias

2.2.7 Selectores nuevos

2.3 Aplicar CSS a nuestro documento

2.3.1 Modelos de caja

2.4 Modelo de caja tradicional

2.4.1 Documento HTML

2.4.2 Selector universal (*)

2.4.3 Títulos

2.4.4 Declaración de nuevos elementos HTML5

2.4.5 Centrar el cuerpo

2.4.6 Creación de la caja principal

2.4.7 La cabecera

2.4.8 Barra de navegación

2.4.9 Área principal y Barra lateral

2.4.10 Pie de página

2.4.11 Toques finales

2.4.12 box-sizing

2.5 Modelo de caja flexible

2.5.1 Contenedor flexible

2.5.2 Documento HTML

2.5.3 Display

2.5.4 Ejes

2.5.5 Propiedad Flex

2.5.6 flex-direction

2.5.7 order

2.5.8 justify-content

2.5.9 align-items

2.5.10 align-self

2.5.11 flex-wrap

2.5.12 align-content

3. Propiedades CSS3

3.1 Las nuevas reglas

3.1.1 CSS3 ha enloquecido

3.1.2 Documento HTML

3.1.3 border-radius

3.1.4 box-shadow

3.1.5 text-shadow

3.1.6 @font-face

3.1.7 linear-gradient

3.1.8 radial-gradient

3.1.9 rgb

3.1.10 hsla

3.1.11 outline

3.1.12 border-image

3.1.13 background

3.1.14 Columnas

3.2 Transformar

3.2.1 transform: scale

3.2.2 transform: rotate

3.2.3 transform: skew

3.2.4 transform: translate

3.2.5 Transformar todo en un elemento

3.2.6 Transformaciones dinámicas

3.2.7 Transformaciones 3D

3.3 Transiciones

3.4 Animaciones

4 Javascript

4.1 Breve introducción a Javascript

4.1.1 El lenguaje

4.1.2 Variables

4.1.3 Condicionales y bucles

4.1.4 Objetos

4.1.5 Constructores

4.1.6 El objeto Window

4.1.7 El objeto Document

4.2 Una introducción a los eventos

4.2.1 Atributos de eventos

4.2.2 Propiedades del evento

4.2.3 El método addEventListener()

4.3 Incorporar Javascript

4.3.1 En línea

4.3.2 Incrustado

4.3.3 Desde un archivo externo

4.4 Nuevos selectores

4.4.1 querySelector()

4.4.2 querySelectorAll()

4.4.3 matchesSelector()

4.5 Interactuar con el documento

4.5.1 Estilos Javascript

4.5.2 ClassList

4.5.3 Acceder a los atributos

4.5.4 dataset

4.5.5 Crear y borrar elementos

4.5.6 innerHTML, outerHtml e insertAdjacentHTML

4.6 Las API

4.6.1 API nativas

4.6.2 API externas

4.7 Errores y depuración

4.7.1 Consola

4.7.2 console.log()

4.7.3 Evento error

5 Formularios

5.1 Formularios HTML

5.1.1 El elemento <form>

5.1.2. El elemento <input>

5.1.3 Más elementos de formulario

5.1.4 Enviar un formulario

5.2 Nuevos tipos de entrada

5.2.1 Tipo email

5.2.2 Tipo search

5.2.3 Tipo url

5.2.4 Tipo tel

5.2.5 Tipo number

5.2.6 Tipo range

5.2.7 Tipo date

5.2.8 Tipo week

5.2.9 Tipo month

5.2.10 Tipo time

5.2.11 Tipo datetime

5.2.12 Tipo datetime-local

5.2.13 Tipo color

5.3 Nuevos atributos

5.3.1 Atributo autocomplete

5.3.2 Atributos novalidate y formnovalidate

5.3.3 Atributo placeholder

5.3.4 Atributo required

5.3.5 Atributo multiple

5.3.6 Atributo autofocus

5.3.7 Atributo pattern

5.3.8 Atributo form

5.4 Nuevos elementos de los formularios

5.4.1 El elemento <datalist>

5.4.2 El elemento <progress>

5.4.3 El elemento <meter>

5.4.4 El elemento <output>

5.5 Nueva pseudo-clases

5.5.1 valid e invalid

5.5.2 optional y required

5.5.3 in-range y out-of-range

5.6 Formularios API

5.6.1 SetCustomValidity()

5.6.2 El evento invalid y el método CheckValidity()

5.6.3 Validación en tiempo real con ValidityState

5.6.4 Restricciones de validez

6 Vídeo y audio

6.1 Vídeo con HTML5

6.1.1 El elemento <video>

6.1.2 Atributos del elemento <video>

6.1.3 Formatos de vídeo

6.2 Audio con HTML5

6.2.1 El elemento <audio>

6.3 Subtítulos

6.3.1 El elemento <track>

6.4 Programar un reproductor multimedia

6.4.1 Diseño de un reproductor de vídeo

6.4.2 Aplicación

6.4.3 Eventos

6.4.4 Código Javascript

6.4.5 Métodos

6.4.6 Propiedades

6.4.7 Código en funcionamiento

7 API TextTrack

7.1 API TextTrack

7.1.1 Lectura de pistas o tracks

7.1.2 Lectura de entradas o cues

7.1.3 Adición de pistas nuevas

8 API Fullscreen

8.1 Basta de ventanas

8.1.1 Ir a pantalla completa

8.1.2 Estilos “Fullscreen”

9. API Stream

9.1 Capturar contenidos

9.1.1 Acceder a la cámara web

9.1.2 Objetos MediaStreamTrack

9.1.3 Método stop()

10 API Canvas

10.1 Los gráficos para la Web

10.1.1 El elemento <canvas>

10.1.2 GetContext()

10.2 Dibujar en el lienzo

10.2.1 Dibujar rectángulos

10.2.2 Color

10.2.3 Degradados

10.2.4 Crear trazados

10.2.5 Estilos de línea

10.2.6 Texto

10.2.7 Sombras

10.2.8 Transformaciones

10.2.9 Restaurar el estado

10.2.10 globalCompositeOperation

10.3 Procesamiento de Imágenes

10.3.1 drawImage()

10.3.2 Datos de imagen

10.3.3 cross-Origin

10.3.4 Extracción de los datos

10.3.5 Patrones

10.4 Animaciones sobre lienzo

10.4.1 Animaciones elementales

10.4.2 Animaciones profesionales

10.5 Procesar vídeo en el lienzo

10.5.1 Mostrar vídeo en el lienzo

10.5.2 Aplicación de la vida real

11. WebGL y Three.js

11.1 Lienzo en 3D

11.2 Three.js

11.2.1 Renderer

11.2.2 scene

11.2.3 Cámara

11.2.4 Mallas

11.2.5 Geométricas primitivas

11.2.6 Materiales

11.2.7 Implementación

11.2.8 Transformaciones

11.2.9 Luces

11.2.10 Texturas

11.2.11 Aplicación UV

11.2.12 Texturas de lienzo

11.2.13 Texturas de vídeo

11.2.14 Cargar modelos 3D

11.2.15 Animaciones en 3D

12 API Pointer Lock

12.1 Nuevo puntero del ratón

12.1.1 Capturar el ratón

12.1.2 pointerLockElement

12.1.3 movementX y movementY

13 API Drag and Drop

13.1 Arrastrar y soltar en la web

13.1.1 Eventos

13.1.2 DataTransfer

13.1.3 dragenter, dragleave y dragend

13.1.4 Seleccionar una fuente válida

13.1.5 setDragImage()

13.1.6 Archivos

14 API Web Storage

14.1 Dos sistemas de almacenamiento

14.2 SessionStorage

14.2.1 Implementar un sistema de almacenamiento de datos

14.2.2 Crear datos

14.2.3 Leer datos

14.2.4 Eliminar datos

14.3 LocalStorage

14.3.1 Evento storage

15 API IndexedDB

15.1 Una API de bajo nivel

15.1.1 Base de datos

15.1.2 Objetos y Almacenes de objetos

15.1.3 Índices

15.1.4 Transacciones

15.1.5 Métodos de almacenes de objetos

15.2 Implementar IndexedDB

15.2.1 Plantilla

15.2.2 Abrir la base de datos

15.2.3 Almacenes de objetos e índices

15.2.4 Agregar objetos

15.2.5 Leer objetos

15.2.6 Finalizar y probar el código

15.3 Listar datos

15.3.1 Cursores

15.3.2 Cambio de orden

15.4 Eliminar datos

15.5 Buscar datos

16 API File

16.1 Almacenamiento de archivos

16.2 Procesar archivos de usuario

16.2.1 Plantilla

16.2.2 Leer archivos

16.2.3 Propiedades de archivos

16.2.4 Blobs

16.2.5 Eventos

16.3 Crear archivos

16.3.1 Documento HTML

16.3.2 El disco duro

16.3.3 Crear archivos

16.3.4 Crear directorios

16.3.5 Listar archivos

16.3.6 Manejar archivos

16.3.7 Mover archivos

16.3.8 Copiar archivos

16.3.9 Eliminar

16.4 Contenido de archivos

16.4.1 Escribir contenido

16.4.2 Agregar contenido

16.4.3 Leer contenido

16.5 Acceder a los archivos

16.6 Sistema de archivos real

17 API Geolocation

17.1 Encontrar su lugar

17.1.1 Documento HTML

17.1.2 getCurrentPosition(ubicación)

17.1.3 getCurrentPosition(ubicación, error)

17.1.4 getCurrentPosition(ubicación, error, configuración)

17.1.5 watchPosition(ubicación, error, configuración)

17.1.6 Usos prácticos con Google Maps

18 API History

18.1 La interfaz de API History

18.1.1 Navegar por la Web

18.1.2 Nuevos métodos

18.1.3 URL falsas

18.1.4 La propiedad state

18.1.5 Un ejemplo de la vida real

19 API Offline

19.1 El manifiesto

19.1.1 El archivo manifiesto

19.1.2 Categorías

19.1.3 Comentarios

19.1.4 Uso del archivo manifiesto

19.2 API Offline

19.2.1 Los errores

19.2.2 online y offline

19.2.3 Estado del caché

19.2.4 Progreso

19.2.5 Actualización del caché

20 API Page Visibility

20.1 El estado de visibilidad

20.1.1 Estado actual

20.1.2 Una mejor experiencia

20.1.3 Detector completo

21 Ajax Level 2

21.1 XMLHttpRequest

21.1.1 Recuperar datos

21.1.2 Propiedades de respuesta

21.1.3 Eventos

21.1.4 Envío de datos

21.1.5 Solicitudes de orígenes cruzados

21.1.6 Cargar archivos

21.1.7 Una aplicación real

22 API Web Messaging

22.1 Mensajería entre documentos

22.1.1 Enviar un mensaje

22.1.2 Comunicar con un iframe

22.1.3 Filtros y orígenes cruzados

23 API WebSocket

23.1 WebSockets

23.1.1 Servidor WebSocket

23.1.2 Instalación y ejecución de un servidor WS

23.1.3 Constructor

23.1.4 Métodos

23.1.5 Propiedades

23.1.6 Eventos

23.1.7 Documento HTML

23.1.8 Iniciar la comunicación

23.1.9 Aplicación completa

24 API WebRTC

24.1 Llega la revolución

24.1.1 El viejo paradigma

24.1.2 El nuevo paradigma

24.1.3 Requisitos

24.1.4 RTCPeerConnection

24.1.5 Candidato ICE

24.1.6 Oferta y respuesta

24.1.7 SessionDescription

24.1.8 Flujos de medios o streams

24.1.9 Eventos

24.1.10 El final

24.2 Ejecutar WebRTC

24.2.1 Servidor de señalización

24.2.2 Servidores ICE

24.2.3 Documento HTML

24.2.4 El código Javascript

24.2.5 Aplicación de la vida real

24.3 Canales de Datos

24.3.1 Creación de canales de datos

24.3.2. Envío de datos

25 API Web Audio

25.1 Estructura de nodos de audio

25.1.1 Los nodos de audio

25.1.2 Contexto Audio

25.1.3 Fuentes de audio

25.1.4 Nodos de conexión

25.1 Sonidos para la Web

25.2.1 Dos nodos básicos

25.2.2 Bucles y tiempos

25.2.3 Crear AudioNodes

25.2.4 AudioParam

25.2.5 GainNode

25.2.6 DelayNode

25.2.7 BiquadFilterNode

25.2.8 DynamicsCompressorNode

25.2.9 ConvolverNode

25.2.10 PannerNode y sonido 3D

25.12.11 AnalyserNode

26 API Web Workers

26.1 Hacer el trabajo duro

26.1.1 Crear un trabajador

26.1.2 Enviar y recibir mensajes

26.1.3 Detectar errores

26.1.4 Detener el trabajador

26.1.5 API asíncronas

26.1.6 Importación de scripts

26.1.7 Trabajador compartido

Conclusiones

Trabajando para el mundo

Las alternativas

Modernizr

Bibliotecas

Google Chrome Frame

Trabajar para la nube

Las API que no han sido incluidas

Lo que debe saber

Palabras finales del autor

Introducción

HTML5 es mucho más que una nueva versión del viejo lenguaje. Incluso es mucho más que una mejora respecto al lenguaje anterior, que ya podía ser considerado “tecnológicamente antiguo”. Realmente es un nuevo concepto diseñado para la creación de sitios web y aplicaciones en la era de los dispositivos móviles, el cloud computing y las redes.

Todo comenzó hace mucho tiempo con una sencilla versión de HTML que fue desarrollada para crear estructuras básicas de páginas web, organizar contenidos y compartir información. Es preciso recordar que el lenguaje y la misma Web nacieron sobre todo con la intención de comunicar información a través de textos.

El alcance limitado de HTML hizo que las empresas comenzaran a desarrollar lenguajes y aplicaciones que han dotado a la web de nuevas características y han elevado la experiencia del usuario hasta niveles antes inimaginados. Aparecieron poderosos y populares complementos: juegos simples y sencillas animaciones dieron paso a sofisticadas aplicaciones que cambiaron el concepto de la Web para siempre. De todas las opciones propuestas, Java y Flash fueron las más exitosas. Fueron adoptadas masivamente y unánimemente consideradas un sinónimo del futuro de la red. Sin embargo, ante el aumento avasallador del número de usuarios, Internet dejó de ser solo una manera de conectar a amantes de ordenadores para convertirse en un medio indispensable tanto para los negocios como para la interacción social. Y en este contexto, las limitaciones presentes en ambas tecnologías se convirtieron en su sentencia de muerte.

Java y Flash tienen una importante limitación en común y es la falta de integración. Ambas fueron concebidas como complementos: algo que se insertaba en una estructura con la que solo compartían un espacio en la pantalla.

No había comunicación e integración entre aplicaciones y documentos. Esta falta de integración pronto se convirtió en un grave problema que allanó el camino para la evolución de un lenguaje que comparte espacio en el documento con HTML y que no se ve afectado por las limitaciones de sus predecesores.

Javascript, el lenguaje interpretado incorporado en los navegadores, sin duda era la mejor manera de mejorar la experiencia del usuario y proporcionar mayor funcionalidad a la Web. Sin embargo, después de algunos años de intentos fallidos de promoverlo, y ante el mal uso que se hizo del lenguaje, el mercado no llegó a adoptarlo plenamente y su popularidad comenzó a decaer.

Los detractores tenían buenas razones para oponerse a su uso. En ese momento, Javascript no podía reemplazar la funcionalidad de Java y Flash. Y aun cuando era evidente que Java y Flash estaban limitando el alcance de las aplicaciones web y aislaban el contenido de la Web, características como el streaming de vídeo, que se estaban convirtiendo en una parte importante de la web, solo eran ofrecidas de forma efectiva a través de estas tecnologías.

A pesar de su dominio, Java fue decayendo cada vez más. La compleja naturaleza del lenguaje, su lenta evolución y su falta de integración fueron minando su importancia hasta el punto de que hoy en día prácticamente no se utiliza en las aplicaciones web tradicionales. Con Java fuera de juego, los desarrolladores centraron su atención en Flash. Sin embargo, el hecho de que Flash comparte características básicas con su competidor lo hace susceptible a la misma suerte.

Mientras que se desarrollaba esta competencia silenciosa e intensa, el software para acceder a la Web continuaba evolucionando. Junto a las nuevas características y a las técnicas más rápidas de acceso a Internet, los navegadores también estaban mejorando sus motores de Javascript. La mayor potencia trajo consigo nuevas oportunidades que Javascript estaba dispuesto a aceptar.

En algún momento durante este proceso se hizo evidente para algunos desarrolladores que ni Java ni Flash serían capaces de proporcionar las herramientas necesarias para crear las aplicaciones que demandaba un número creciente de usuarios. Estos desarrolladores comenzaron a aplicar Javascript en sus aplicaciones de una manera que no se había visto nunca antes. La innovación y los resultados sorprendentes así obtenidos atrajeron la atención de más programadores y pronto nació la “Web 2.0”. Y a partir de ese momento, la percepción que la comunidad de desarrolladores tenía de Javascript cambió radicalmente.

Javascript es definitivamente el lenguaje que ha permitido a los desarrolladores innovar y crear cosas que nadie había hecho antes en la Web. En los últimos años, a los programadores y diseñadores web de todo el mundo se les han ocurrido métodos increíbles para superar las limitaciones de esta tecnología y sus deficiencias iniciales en cuanto a portabilidad. Pronto se hizo evidente que Javascript, junto a HTML y CSS, es el tercer elemento de una perfecta combinación de lenguajes, necesaria para la evolución de la Web.

HTML5 es el fruto de esa combinación, el pegamento que mantiene todo unido. HTML5 propone normas para todos los aspectos de la Web, así como un propósito claro para cada tecnología involucrada. Actualmente, HTML proporciona los elementos estructurales, CSS se encarga de la presentación para hacerlos más atractivo y útiles, y Javascript tiene el poder necesario para proporcionar la funcionalidad y crear aplicaciones web completas.

Los límites entre sitios web y aplicaciones finalmente han desaparecido. Las tecnologías necesarias están listas. La promesa es el futuro de la Web, y la evolución y la combinación de estas tres tecnologías (HTML, CSS y Javascript) es una alianza de gran alcance que ha convertido a Internet en la plataforma líder para el desarrollo. HTML5 está claramente a la cabeza.

Importante

Al momento de preparar este libro, no todos los navegadores soportan de manera absoluta las características de HTML5 y la mayoría de éstas son aún experimentales. Le recomendamos que a medida que lea sus lecciones, ejecute el código con la última versión de Google Chrome y Mozilla Firefox.

Google Chrome es una buena plataforma de pruebas. Está basado en un motor del navegación de código abierto (WebKit) y es compatible con casi todas las características ya implementadas en HTML5. Por otro lado, Mozilla Firefox es uno de los mejores navegadores para los desarrolladores y también proporciona soporte para HTML5. Los ejemplos de este libro fueron preparados para estos dos navegadores, pero para simplificar, usted encontrará algunos scripts que ejecutan solo los métodos experimentales proporcionados por Google Chrome. Por esta razón, le recomendamos estudiar los códigos en Google Chrome en primer lugar y luego expandir su trabajo a otros navegadores.

Sin importar cual sea el navegador que utilice, siempre tenga en cuenta que un buen desarrollador instala y prueba sus códigos en cada programa disponible en el mercado, así que compruebe los códigos proporcionados en este libro con todos los navegadores. Para descargar las versiones más recientes, visite los siguientes sitios web:

www.google.com/chrome

www.apple.com/safari/download

www.mozilla.com

windows.microsoft.com

www.opera.com

www.maxthon.com

1. Documentos HTML5

1.1 Componentes básicos

El lenguaje HTML5 proporciona tres características básicas: estructura, estilo y funcionalidad. Nunca fue declarado oficialmente pero, aun cuando algunas API (Interfaces de programación de apliaciones) de Javascript y la especificación completa CSS3 no forman parte del lenguaje, HTML5 es considerado un producto que surge de la combinación de HTML, CSS y Javascript. Estas tecnologías son altamente confiables y actúan como una unidad organizada de acuerdo a la especificación HTML5. HTML es el encargado de la estructura, CSS presenta esa estructura y su contenido en la pantalla, y Javascript hace el resto, que (como veremos más adelante en este libro) es muy importante.

Importante

Para acceder a información adicional y a la lista de ejemplos, visite nuestro sitio web en www.minkbooks.com.

A pesar de la integración de estas tecnologías, la estructura sigue siendo la parte fundamental de un documento. Proporciona los elementos necesarios para distribuir el contenido estático o dinámico, y es también una plataforma básica para las aplicaciones.

Con la variedad de dispositivos que acceden a Internet y la diversidad de interfaces que se utilizan para interactuar con la Web, un aspecto tan básico como la estructura se convierte en una parte vital del documento. Ahora bien, la

estructura debe proporcionar forma, organización y flexibilidad, y debe ser tan sólida como los cimientos de una casa.

Para trabajar con HTML5, ya sea en la creación de páginas web o de aplicaciones, primero necesitamos saber cómo se construye esa estructura. La creación de una base sólida nos ayudará después aplicar otros componentes para sacar el máximo provecho de estas nuevas posibilidades. Por ello vamos a empezar con lo básico, paso a paso. En este primer capítulo, aprenderá lo que es HTML y cómo construir una estructura esencial utilizando los nuevos elementos de este lenguaje.

1.2 Una breve introducción a HTML

El HyperText Markup Language (HTML) es un lenguaje de programación. A diferencia de otros lenguajes, no está compuesto por instrucciones, sino por un conjunto de etiquetas que organizan y declaran el propósito de cada contenido del documento.

En el sentido estricto, HTML es un texto escrito con una sintaxis particular que el navegador es capaz de leer y aplicar. Se trata de un lenguaje que fue creado para compartir a través de Internet no solo el texto incluido en los documentos, sino también su formato.

Precisamente es esta posibilidad de diferenciar las partes importantes del contenido de un documento proporcionada por el lenguaje HTML lo que ha abierto la puerta a la creación de la Web como la conocemos hoy en día.

1.2.1 Etiquetas y elementos

Efectivamente, el código HTML no es un conjunto secuencial de instrucciones sino es un lenguaje de marcado, un conjunto de etiquetas que por lo general vienen en pares y que pueden ser anidadas (contenidas dentro de otros elementos). Estas etiquetas son palabras clave y atributos encerrados entre corchetes angulares (por ejemplo, <html lang=“en”>). Por lo general, nos referimos a una etiqueta individual simplemente como “etiqueta” y a un par de etiquetas de apertura y cierre como un “elemento”. Observe el siguiente ejemplo:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi primer documento HTML</title>
</head>
<body>
  ¡Buenos días!
</body>
</html>
```

Código 1-1

Ejemplo de un documento básico de HTML.

Importante

Los conceptos que aquí se explican son básicos pero esenciales para la comprensión de los ejemplos en este libro. Si está familiarizado con esta información, por favor siéntase con la libertad de saltar sobre los temas que ya domina.

Hágalo usted mismo

Un documento HTML es un archivo de texto. Si no dispone de ningún software de desarrollo para crear el archivo, puede utilizar simplemente cualquier editor de texto como el Notepad de Windows. Cree un nuevo archivo con el Código 1-1, guárdelo con un nombre y la extensión .html (por ejemplo, prueba.html) y a continuación abra el archivo en el navegador. Si su editor de texto trabaja con diferentes formatos, asegúrese de guardar el archivo como texto sin formato antes de cambiar su extensión. Para ejecutar el archivo en el navegador configurado por defecto en su ordenador bastará con hacer doble clic sobre él en el explorador de archivos.

Algunos elementos de HTML son simples, es decir, están compuestos de una sola etiqueta, pero la mayoría requiere una etiqueta de apertura y otra de cierre. En el ejemplo del Código 1-1 podemos ver varias etiquetas, una tras otra, con texto e incluso con otras etiquetas en medio. Compare las etiquetas de apertura y de cierre de este ejemplo, y verá que la de cierre se distingue por tener una barra antes de la palabra clave. Por ejemplo, la etiqueta <html> indica el inicio del código HTML, mientras que </html> declara el final. El navegador tratará todo lo que se encuentre en medio de estas etiquetas como código HTML.

Importante

No se preocupe si no entiende cómo se crea la estructura presentada en el Código 1-1. Vamos a trabajar en ello y en los nuevos elementos estructurales introducidos en HTML5 en las próximas páginas.

En este ejemplo, aunque se trata de un código muy sencillo, ya podemos ver una estructura compleja. En la primera línea, hay una sola etiqueta con la definición del documento seguido de la etiqueta de apertura <html lang=“en”>. Entre las etiquetas <html> insertamos otras etiquetas, como las que representan a la cabeza y el cuerpo del documento (<head> y <body>), que también vienen en pares y encierran contenido adicional, como texto o de otros elementos (<title>).

Como se puede ver, para construir un documento HTML, las etiquetas aparecen una tras otra y también entre otras etiquetas, lo que da como resultado en una estructura en árbol, en la que la etiqueta <html> es la raíz.

En general, todo elemento puede ser anidado, ser un contenedor o ser contenido por otros elementos. Básicamente, elementos estructurales especiales como

<head>, <html> o <body> tienen un lugar específico en un documento HTML, pero el resto de elementos son flexibles, como veremos más adelante en este mismo capítulo.

1.2.2 Atributos

Como podrá darse cuenta, la etiqueta de apertura <html> del Código 1-1 no solo se compone de la palabra clave y el corchete angular, sino también de la cadena de texto lang=“es”. Se trata de un atributo con un valor. El nombre del atributo es lang y mediante el símbolo = se le ha asignado el valor “es”. Los atributos proporcionan información adicional acerca de un elemento HTML. En este caso, se declara que el lenguaje humano del código HTML es el español.

Los atributos siempre se declaran dentro de la etiqueta de apertura y pueden tener una estructura nombre=valor, tal como sucede con el atributo lang de nuestro ejemplo, o representar un valor booleano, en cuyo caso su presencia indica la condición verdadera o falsa (por ejemplo, el atributo disabled sin un valor específico deshabilita un elemento de formulario).

Dado que el lenguaje HTML pasó de tener un propósito general para centrarse más en la creación de la estructura del documento, la mayoría de los atributos clásicos han dejado de usarse y algunos han sido sustituidos incluso completamente por las propiedades CSS, como veremos en los capítulos siguientes. Sin embargo, algunos siguen siendo útiles; especialmente cuatro atributos genéricos que tienen especial importancia en el desarrollo de sitios y aplicaciones web en HTML5:

class: Este atributo permite trabajar con un grupo de elementos que comparten algunas características. Por ejemplo, es posible asignar el mismo tipo de fuente y estilos a textos ubicados en diferentes partes del documento simplemente asignando un valor para el atributo class.

id: Este atributo nos permite asignar un identificador único para cada elemento. Es la mejor manera de tener acceso a partir de CSS o Javascript a un elemento específico.

style: Este atributo nos permite asignar estilos CSS para cada elemento, de forma individual. Como veremos en el capítulo 2, es recomendable evitar el uso de este atributo y, en cambio, asignar estilos a los elementos HTML haciendo referencia a sus atributos class e id.

name: Es un viejo atributo que aún tiene aplicaciones cuando trabajamos con formularios. Simplemente declara un nombre personalizado para el elemento. Para obtener más información, consulte el Capítulo 5.

Además de los mencionados, hay atributos más específicos, como el atributo lang del elemento <html> del que también ya hemos hablado. Otros dos que vale la pena mencionar son el atributo href para el elemento <link> y el elemento <a>, así como el atributo src para elementos multimedia como son <img>, <video>, <audio>, etc, que indican la ruta para cargar el archivo o acceder a él.

Pruebe el ejemplo siguiente en su navegador:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title> Mi segundo documento HTML </title>
</head>
<body>
  <img src="http://www.minkbooks.com/content/myimage.jpg">
</body>
</html>
```

Código 1-2

Uso del elemento <img> y del atributo src para mostrar una imagen.

Importante

Este libro se centra en las mejoras introducidas por HTML5. Los conceptos del lenguaje HTML que fueron introducidos antes de HTML5 se explican solo en las secciones denominadas Conceptos básicos. Si desea realizar un estudio más profundo del lenguaje HTML y encontrar una lista completa de los elementos HTML válidos para ser utilizados en proyectos, por favor visite nuestro sitio web y siga los enlaces dispuestos para este capítulo.

Conceptos básicos

El elemento <img> permite cargar y mostrar una imagen en la pantalla. Se trata de un elemento único sin etiqueta de cierre, y se usa junto con el atributo src para declarar la ruta de la imagen. También es posible declarar los atributos width y height para establecer el tamaño de la imagen. Vamos a estudiar y experimentar con estos y otros atributos en situaciones más prácticas a lo largo del libro.

1.2.3 Elementos anteriores

En HTML5 algunos elementos han quedado obsoletos porque algunos elementos han cambiado y se han añadido otros nuevos. Entre ellos, hay algunos elementos que ya no se consideran parte del lenguaje, como , <frame> o <font>. El último es tal vez el más importante. El elemento <font>, ahora generalmente sustituido por el elemento , se utilizó hasta HTML4 para mostrar texto en la pantalla.

Otros elementos como <i> o <b>, que antes eran utilizados para enfatizar texto en pantalla, ahora tienen un significado diferente. Al mismo tiempo, nuevos elementos como <mark>, <address> o <time>, por ejemplo, fueron añadidos para proporcionar una mejor forma de describir y representar el contenido del documento.

1.3 Estructura global

Los documentos HTML están estrictamente organizados. Cada parte del documento está diferenciada, declarada y encerrada entre etiquetas específicas. En esta sección vamos a ver cómo construir la estructura global de un documento HTML y cómo ha cambiado este proceso en HTML5.

Hágalo usted mismo

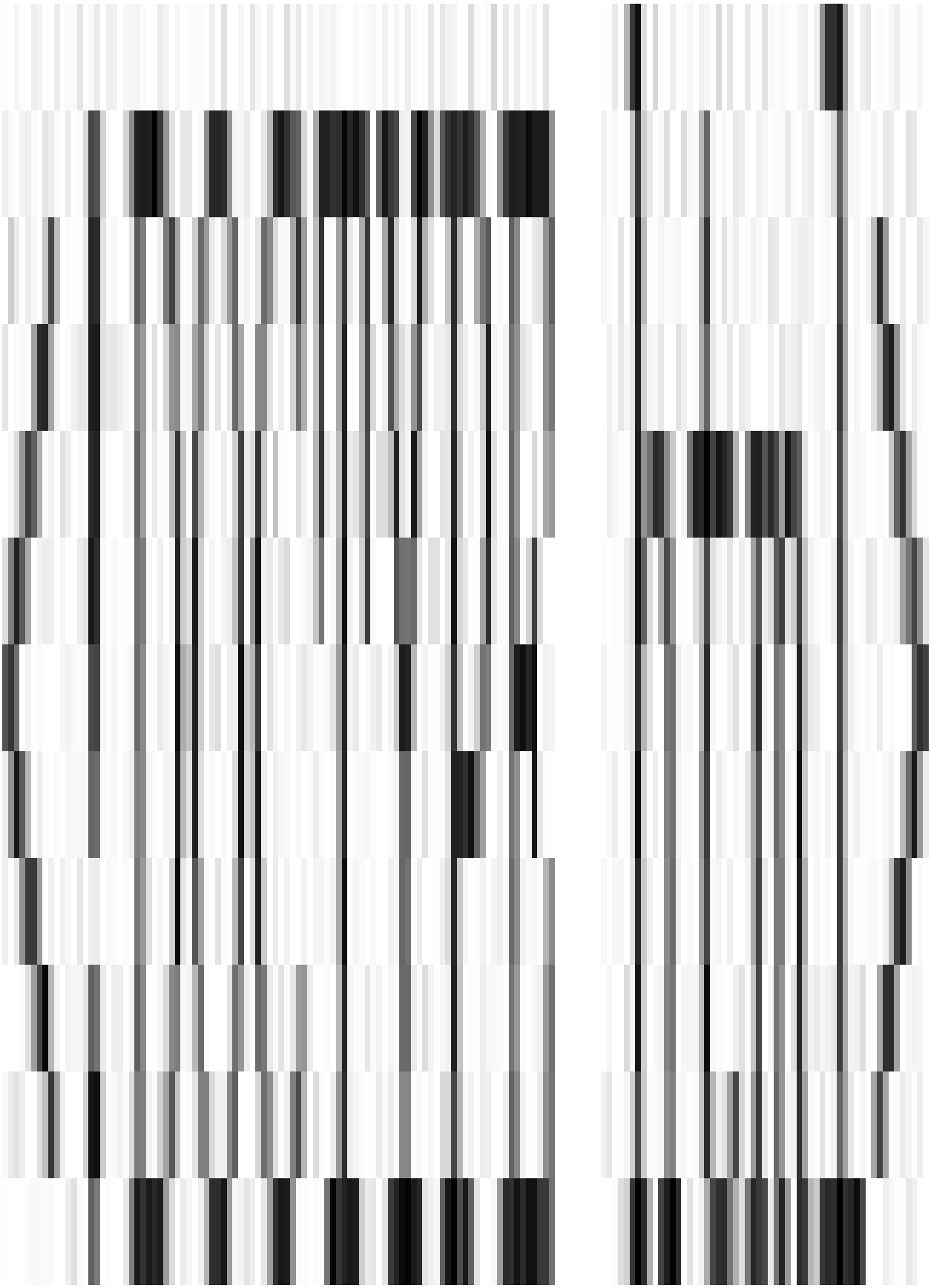
Cree un nuevo documento en un editor de texto para probar en su navegador el código HTML que se presenta más adelante. Esto le ayudará a recordar las nuevas etiquetas y a familiarizarse con su uso.

Importante

Los navegadores ofrecen estilos mínimos para los elementos HTML. Para ver la estructura creada con código HTML en este capítulo en la pantalla, tendrá que aplicar estilos CSS. Estudiaremos el uso de CSS en los capítulos 2 y 3.

1.3.1 <!DOCTYPE>

En primer lugar, es necesario que indicar el tipo de documento que se va a crear, porque los navegadores son capaces de procesar diferentes tipos de archivos. Para asegurarse de que el documento sea interpretado como código HTML de forma correcta, es necesario declarar el tipo de documento al principio del mismo. En HTML5 es simple:



Código 1-3

Uso del elemento <! DOCTYPE>.

Tenga en cuenta que esta línea debe ser la primera línea de su archivo y que no debe haber espacios ni líneas antes de ella. De esta forma se activa un modo y se fuerza a los navegadores a interpretar HTML5 si es posible, o ignorarlo si no es el caso.

Hágalo usted mismo

Puede comenzar a escribir el Código 1.3 en un archivo HTML e ir añadiendo cada uno de los nuevos elementos que presentaremos en las próximas páginas.

1.3.2 <html>

Después de declarar el tipo de documento, tenemos que construir la estructura de árbol del HTML. Como siempre, el elemento raíz de este árbol es <html>. Éste es el elemento que contendrá todo el código HTML.

<!doctype html>
<html lang="es">
</html>

Código 1-4

Uso del elemento <html>.

El atributo lang en la etiqueta de apertura <html> es el único que necesitará indicar en HTML5. Recuerde que hemos dicho que este atributo define el idioma que se utiliza para el contenido del documento, que en este caso es, obviamente, español.

Para conocer otros idiomas para el atributo lang, puede seguir este enlace: http://www.w3schools.com/tags/ref_language_codes.asp

Importante

HTML5 es muy flexible en cuanto a la estructura y a los elementos utilizados para su construcción. El elemento <html> también puede ser incluido sin ningún atributo o incluso puede ser ignorado. Sin embargo, por razones de compatibilidad y otros motivos en los que no vale la pena extendernos en este manual, le recomendamos que siga nuestras reglas básicas. Tenga en cuenta que le explicaremos cómo crear documentos HTML de acuerdo con las que consideramos que son las mejores prácticas.

1.3.3 <head>

Continuemos avanzando en la construcción del documento. El código HTML insertado entre las etiquetas <html> ha de estar dividido en dos secciones principales. Tal como sucedía en versiones anteriores de HTML, la primera sección es la cabeza (head) y la segunda el cuerpo (body). Así que el próximo paso será la creación de estas dos secciones en el código utilizando las etiquetas <head> y <body>.

Evidentemente, el elemento <head> va primero y, como el resto de los elementos estructurales, tiene una etiqueta de apertura y una etiqueta de cierre.

```
<!DOCTYPE html>

<html lang="es">

<head>

</head>

</html>
```

Código 1-5

Uso del elemento <head>.

La etiqueta en sí no ha cambiado respecto a versiones anteriores y su propósito también sigue siendo exactamente el mismo. Dentro de las etiquetas <head> vamos a definir el título de nuestra página web, a declarar la codificación de caracteres, a proporcionar información general sobre el documento y a incorporar archivos externos con los estilos, los scripts e incluso las imágenes necesarias para identificar la página. A excepción del título y de algunos iconos, la información incorporada al documento entre las etiquetas <head> normalmente no está a la vista del usuario.

1.3.4 <body>

La siguiente sección en un documento HTML contiene la parte visible del documento y se identifica con la etiqueta <body>. Se trata, nuevamente, de una etiqueta que no ha cambiado respecto a las versiones anteriores de HTML.

```
<!DOCTYPE html>
<html lang="es">
<head>
</head>
<body>
</body>
</html>
```

Código 1-6

Uso del elemento <body>.

1.3.5 <meta>

A continuación construiremos el elemento head del documento. El contenido de este elemento ha sufrido algunos cambios y uno de ellos es la etiqueta que define la codificación de caracteres del documento, que es la etiqueta meta, que indica cómo debe presentarse el texto en la pantalla.

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="utf-8">
</head>
<body>
</body>
</html>
```

Código 1-7

Uso del elemento <meta>.

La novedad de este elemento en HTML5, como en la mayoría de los casos, es la simplificación alcanzada. La nueva etiqueta <meta> para la codificación de caracteres es más corta y más simple. Por supuesto, puede utilizar la codificación que prefiera en lugar de utf-8 y además puede añadir las etiquetas <description> o <keywords>, tal como mostramos en el ejemplo siguiente.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <meta name="description" content="Este es un ejemplo de HTML5">
  <meta name="keywords" content="HTML5, CSS3, Javascript">
</head>
<body>
</body>
</html>
```

Código 1-8

Uso de otros elementos <meta>.

Conceptos básicos

Hay varias etiquetas <meta> que pueden ser utilizadas en un documento para declarar la información general que no será mostrada en la ventana del navegador. Se trata de información que es utilizada por los motores de búsqueda y los dispositivos que necesitan obtener una vista previa o conseguir un resumen de los datos relevantes del documento. En el Código 1-8, el atributo name dentro de la etiqueta <meta> especifica el tipo y el atributo content declara su valor, pero ninguno de estos valores se muestra en la pantalla. Para obtener más información sobre las etiquetas <meta>, visite nuestro sitio web y siga los enlaces de este capítulo.

Debe saber que en HTML5, no es necesario cerrar las etiquetas individuales con una barra al final, pero es recomendable hacerlo por razones de compatibilidad. A continuación, añadimos al código anterior barras inclinadas:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8" />
  <meta name="description" content="Éste es un ejemplo de HTML5" />
  <meta name="keywords" content="HTML5, CSS3, JavaScript" />
</head>
<body>
</body>
</html>
```

Código 1-9

Etiquetas con barras de cierre.

1.3.6 <title>

La etiqueta <title>, tal como sucedía en versiones anteriores, simplemente indica el título del documento, de manera que no hay ninguna novedad que comentar sobre la misma.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <meta name="description" content="Éste es un ejemplo de HTML5">
  <meta name="keywords" content="HTML5, CSS3, JavaScript">
  <title>Éste es el título del documento</title>
</head>
<body>
</body>
</html>
```

Código 1-10

Uso del elemento <title>.

Conceptos básicos

El texto que se encuentra entre las etiquetas <title> es el título del documento que estamos creando. Por lo general, los navegadores muestran este texto en la parte superior de la ventana.

1.3.7 <link>

Otra parte importante del elemento <head> o la cabeza de un documento HTML5 es el elemento <link>. Este elemento se utiliza para incorporar estilos, scripts, imágenes o iconos desde archivos externos en el documento. Uno de sus usos más habituales es la incorporación de estilos mediante la inserción de un archivo CSS externo.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <meta name="description" content="Éste es un ejemplo de HTML5">
  <meta name="keywords" content="HTML5, CSS3, JavaScript">
  <title>Éste es el título del documento</title>
  <link rel="stylesheet" href="misestilos.css">
</head>
<body>
</body>
</html>
```

Código 1-11

Uso del elemento <link>.

Como en el lenguaje HTML5 no es necesario indicar el tipo de hoja de estilo, el atributo type ha sido eliminado. Ahora solo son necesarios dos atributos para incorporar la hoja de estilo: rel y href. El nombre del atributo rel significa relación y establece la relación entre el documento y el archivo incorporado. En este caso, el atributo rel tiene el valor stylesheet que indica al navegador que el archivo misestilos.css es un archivo CSS que contiene los estilos necesarios para interpretar la página.

Importante

En el Capítulo 2 se estudiará el lenguaje CSS y el proceso de creación del archivo misestilos.css para definir el estilo del documento.

El atributo href, como hemos explicado antes, declara la ruta para cargar el archivo. Este archivo, por supuesto, debe tener un contenido adecuado al valor del atributo rel. En este caso, la ruta apunta a un archivo CSS que contiene los estilos para el documento (stylesheet).

Conceptos básicos

Una hoja de estilo (stylesheet) es un conjunto de reglas de formato que modifican el aspecto del documento, por ejemplo, el tamaño y el color del texto. Sin estas reglas, el texto y otros elementos se mostrarán en la pantalla según los estilos estándar proporcionados por el navegador (tamaños predeterminados, colores, etc).

Como verá más adelante, los estilos son simples reglas y por lo general requieren solo unas pocas líneas de código que también pueden ser declaradas en el mismo documento, así que no es estrictamente necesario obtener esta información a partir de archivos externos, pero sí que es una práctica recomendable porque permite organizar el documento principal, aumentar la velocidad de carga del sitio web y aprovechar las nuevas características HTML5.

Con la última instrucción insertada, podemos dar por terminada la cabeza de nuestro documento. Ahora estamos en condiciones de agregar contenido a las etiquetas <body> para que comience la magia.

1.4 La estructura del cuerpo del documento

La estructura del cuerpo del documento, que es el código que se encuentra entre las etiquetas <body>, es la que generará la parte visible del mismo: el código que realmente le dará forma a la página web.

El lenguaje HTML siempre ha ofrecido distintas formas de construir y organizar la información visible en el cuerpo del documento. Uno de los primeros elementos utilizados para este fin era el elemento <table>, que permitía a los autores organizar datos, textos, imágenes y herramientas en filas y columnas de celdas, a pensar de que no había sido concebido para este fin.

El elemento <table> representó una de las primeras revoluciones de la web, un gran paso adelante en la visualización del documento que mejoró la experiencia de los usuarios. Con el tiempo otros elementos sustituyeron la función de este elemento pues proporcionaban una manera diferente de hacer lo mismo, pero más rápido y con menos códigos, facilitando así la creación, el mantenimiento y la portabilidad el sitio.

El elemento fue uno de los que adquirió gran importancia. Con la aparición de nuevas aplicaciones web interactivas y la integración de HTML, CSS y Javascript, su uso se convirtió en una práctica habitual. Pero al igual que sucedía con el elemento <table>, el elemento no proporciona mucha información sobre las partes del cuerpo del documento que representa. Cualquier cosa, desde imágenes hasta menús, textos, enlaces, scripts, formularios, etc., puede ir entre las etiquetas de apertura y cierre. En otras palabras, la palabra clave solo establece una división en el cuerpo, como las celdas de una tabla, pero no da una idea de qué tipo de división que es o cuál es el propósito de esa división y de su contenido.

Esta información, que es irrelevante para los usuarios, resulta en cambio de una importancia crucial para que los navegadores puedan realizar una correcta interpretación del contenido. Tras la revolución de los nuevos dispositivos móviles y las nuevas formas de acceder a la web, esta correcta identificación de cada una de las partes de una web ha adquirido una enorme relevancia.

Considerando estas circunstancias, HTML5 ha incorporado nuevos elementos que ayudan a identificar cada sección del documento y a organizar el cuerpo del mismo. En HTML5 las secciones más importantes se encuentran por tanto claramente diferenciadas y la estructura principal ya no está definida por las etiquetas y <table>.

1.4.1 Organización

La Figura 1-1 representa un diseño regular que actualmente es utilizado en la mayoría de los sitios web. A pesar del hecho de que cada diseñador crea sus propios diseños, en general, podremos identificar las siguientes secciones en prácticamente cualquier sitio web:

Cabecera
Barra de navegación
Información principal
Barra lateral
Institucional

Figura 1-1

Representación visual del diseño de una típica página web.

En la parte superior, que hemos llamado Cabecera del sitio, normalmente se encuentran el logo, el título o nombre de la página, los subtítulos y una breve descripción del sitio o página web.

La mayoría de los programadores ubican a continuación una Barra de navegación en la que ofrecen un menú o una lista de enlaces para facilitar la navegación, que dirige a los usuarios a diferentes páginas o documentos, por lo general en el mismo sitio web.

El contenido más relevante de la página se coloca generalmente en el centro de ésta. En esta sección se presenta la información más importante además de los enlaces y habitualmente está dividida en filas y columnas. En el ejemplo de la Figura 1-1 se pueden ver solo dos columnas: la Información principal y la Barra lateral, pero sepa que se trata de una sección extremadamente flexible, que los diseñadores suelen adaptar a según sus necesidades mediante la inserción de más filas, la división de las columnas en bloques más pequeños o la generación de diferentes combinaciones y distribuciones.

El contenido que se presenta en esta parte del diseño generalmente tiene la mayor prioridad o relevancia. En el ejemplo que presentamos, la Información principal podría incluir una lista de artículos, descripciones de productos, entradas de blog, o cualquier otro contenido relevante, y la Barra lateral podría mostrar una lista de enlaces relacionados con esos elementos. En un blog, por ejemplo, esta última columna ofrece una lista de enlaces a las entradas del blog,

la información sobre el autor, etc.

En la parte inferior de la disposición típica, se encuentra el pie o Barra institucional. Le hemos dado este nombre porque suele mostrar información general sobre el sitio web, el autor, la compañía, además de enlaces a reglas, términos y condiciones, mapas y toda clase de datos adicionales sobre el promotor. La Barra institucional es el complemento de la Cabecera y es considerada actualmente una parte esencial de la estructura de una página web.

La Figura 1-2 es la representación de un blog estándar y en ella donde se pueden identificar claramente las partes del diseño descritas anteriormente:

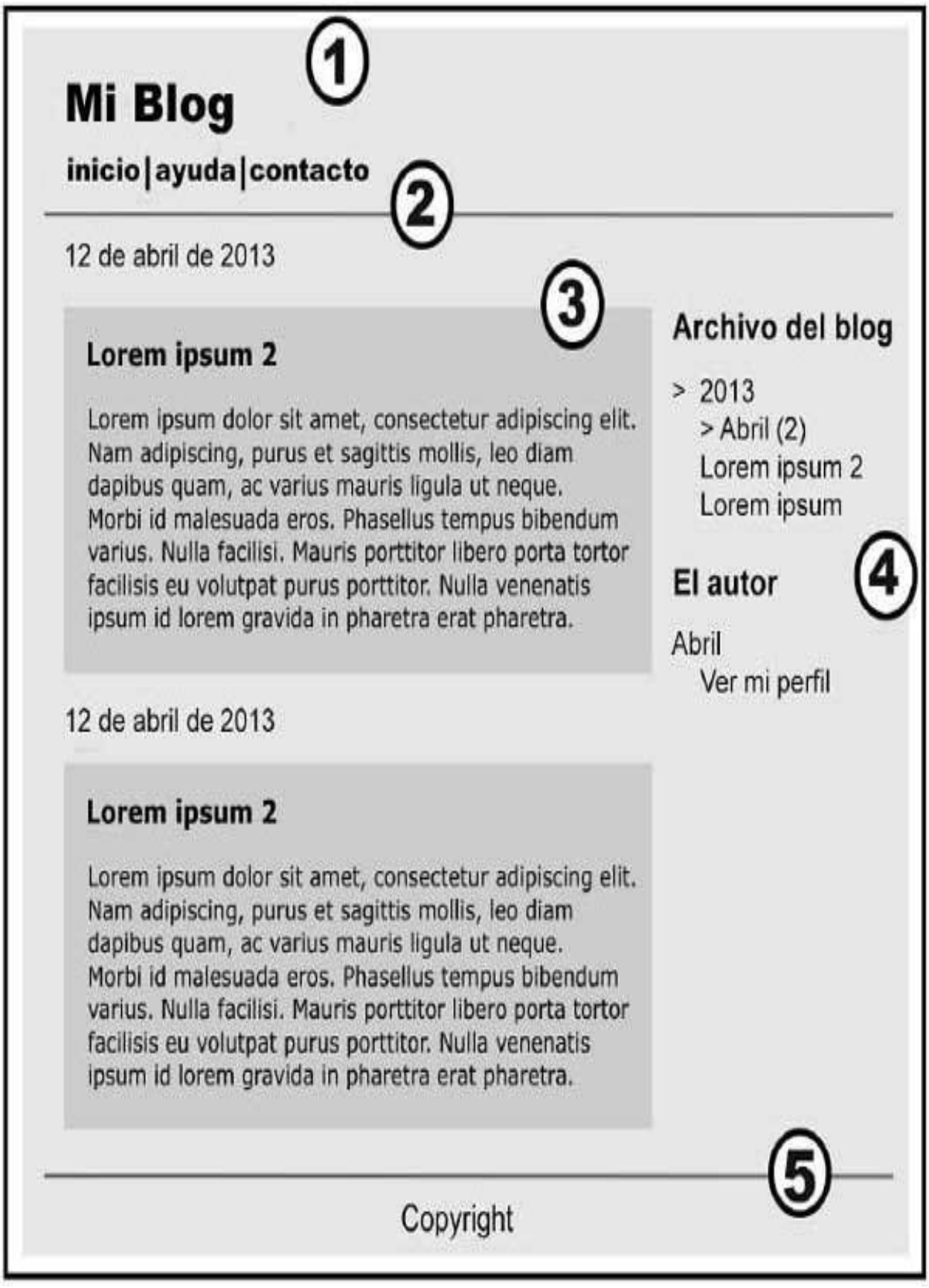

1
Mi Blog
inicio|ayuda|contacto
2
12 de abril de 2013
3
Lorem ipsum 2
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nam adipiscing, purus et sagittis mollis, leo diam dapibus quam, ac varius mauris ligula ut neque. Morbi id malesuada eros. Phasellus tempus bibendum varius. Nulla facilisi. Mauris porttitor libero porta tortor facilisis eu volutpat purus porttitor. Nulla venenatis ipsum id lorem gravida in pharetra erat pharetra.
Archivo del blog
> 2013
> Abril (2)
Lorem ipsum 2
Lorem ipsum
El autor
4
Abril
Ver mi perfil
12 de abril de 2013
Lorem ipsum 2
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nam adipiscing, purus et sagittis mollis, leo diam dapibus quam, ac varius mauris ligula ut neque. Morbi id malesuada eros. Phasellus tempus bibendum varius. Nulla facilisi. Mauris porttitor libero porta tortor facilisis eu volutpat purus porttitor. Nulla venenatis ipsum id lorem gravida in pharetra erat pharetra.
5
Copyright

Figura 1-2

Estructura de un blog estándar.

1. Cabecera

2. Barra de navegación

3. Información principal

4. Barra lateral

5. Pie o Barra institucional

Esta sencilla representación de un blog que acaba de ver muestra claramente que cada sección definida en un sitio web tiene un propósito. Aunque el propósito no siempre será tan explícito, siempre se mantiene la esencia y cualquier usuario será capaz de reconocer estos elementos en cualquier sitio.

HTML5 parte de esta estructura básica de diseño, y proporciona nuevos elementos para diferenciar y declarar cada uno de ellos. Ahora podemos anunciar a los navegadores cuál es el objetivo de cada una de las secciones. La Figura 1-3 muestra la disposición típica de la que venimos hablando, pero esta

vez con los elementos de HTML5 correspondientes para cada sección (tanto etiquetas de apertura como de cierre).

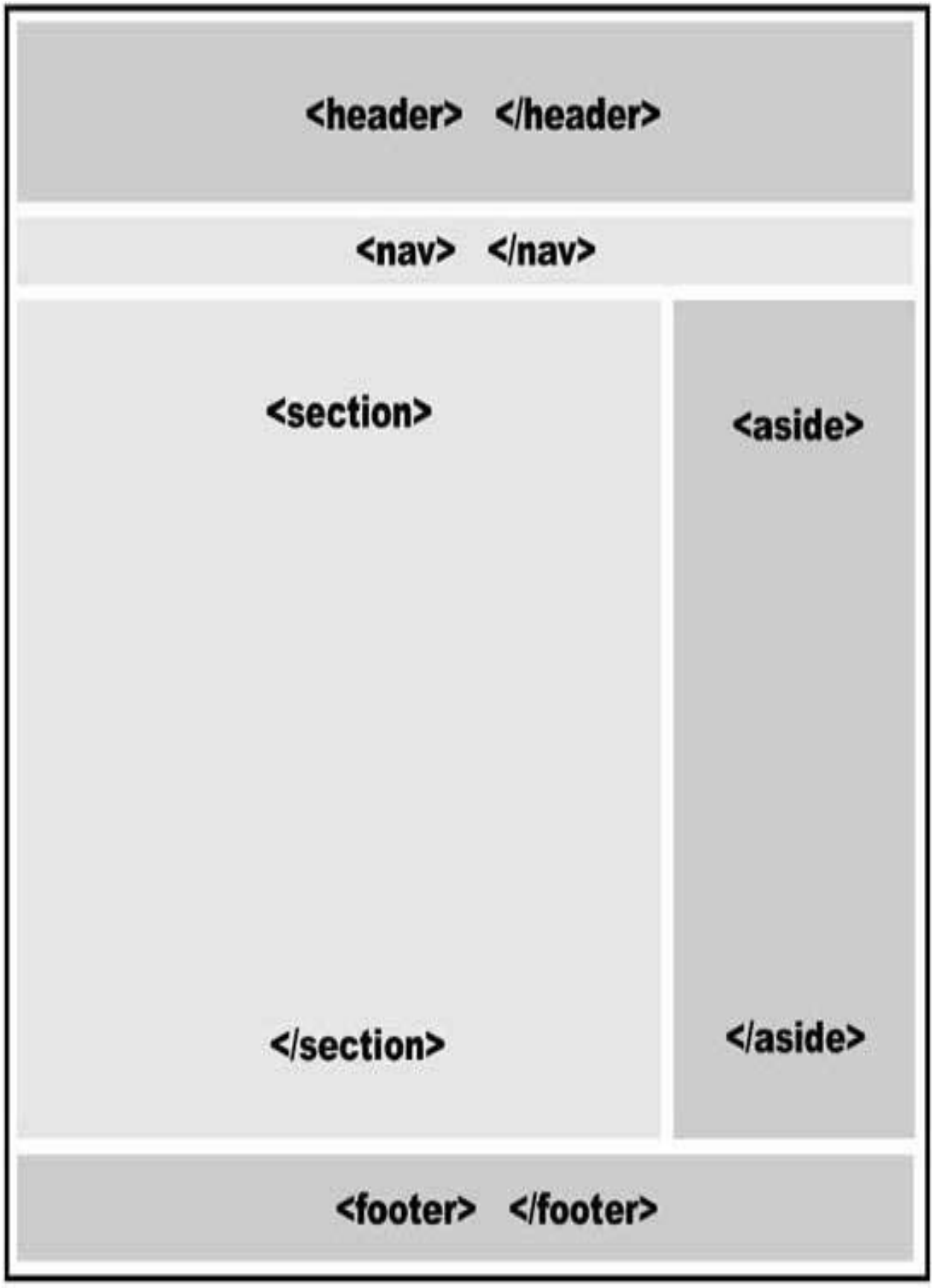

<header> </header>
<nav> </nav>
<section>
<aside>
</section>
</aside>
<footer> </footer>

Figura 1-3

Representación visual de la organización de un documento con las etiquetas de HTML5.

1.4.2 <header>

Uno de los nuevos elementos incorporados en HTML5 es el llamado <header>. El <header> no debe confundirse con la etiqueta <head>, que usamos páginas atrás para construir la cabeza del documento. De la misma manera que la etiqueta <head>, la etiqueta <header> proporciona informaciones introductorias (tales como títulos, subtítulos o logotipos), pero ambas etiquetas difieren en alcance. Si bien las etiquetas <head> tienen el propósito de proporcionar información sobre todo el documento, las etiquetas <header> están destinadas a ser utilizadas como parte del cuerpo del documento HTML.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <meta name="description" content="Éste es un ejemplo de HTML5">
  <meta name="keywords" content="HTML5, CSS3, JavaScript">
  <title>Éste es el título del documento</title>
  <link rel="stylesheet" href="misestilos.css">
</head>
<body>
  <header>
    <h1>Título principal de la página</h1>
  </header>
</body>
</html>
```

Código 1-12

Uso del elemento <header>.

En el Código 1-12 definimos el título de la página web con la etiqueta <header>. Tenga en cuenta que este título es distinto al título general del documento que definimos anteriormente en la sección <head>. La inserción del elemento de <header> representa el comienzo del cuerpo del documento y, por tanto, de su parte visible. A partir de este punto será capaz de ver los resultados del código que está creando en la ventana de su navegador.

Hágalo usted mismo

Si ha seguido las instrucciones desde el comienzo de este capítulo, ya debe disponer de un archivo de texto con los códigos HTML estudiados hasta ahora, listo para ser probado. Si no, lo único que tiene que hacer es copiar el contenido del Código 1-12 en un archivo de texto vacío creado con cualquier editor de texto (como el Bloc de notas de Windows), guardar el archivo con un nombre y la extensión .html, y abrirlo en su navegador.

Conceptos básicos

El elemento <h1>, aplicado en el Código 1-12, es un viejo elemento del lenguaje

HTML utilizado para definir encabezados. En número, que puede estar comprendido entre el 1 y el 6, indica la importancia del encabezado. El elemento <h1> es el de mayor importancia y el elemento <h6> el de menor importancia, de manera que <h1> será el adecuado para mostrar el título principal y el resto de los subtítulos. Posteriormente, se estudiará cómo ha sido modificada la aplicación de estos elementos en HTML5.

1.4.3 <nav>

La siguiente sección de nuestro ejemplo es la barra de navegación, que en HTML5 se genera con la etiqueta <nav>:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <meta name="description" content="Éste es un ejemplo de HTML5">
  <meta name="keywords" content="HTML5, CSS3, JavaScript">
  <title>Éste es el título del documento</title>
  <link rel="stylesheet" href="misestilos.css">
</head>
<body>
  <header>
    <h1>Título principal de la página</h1>
  </header>
  <nav>
    <ul>
      <li>Inicio</li>
      <li>Imágenes</li>
      <li>Vídeos</li>
      <li>Contacto</li>
    </ul>
  </nav>
</body>
</html>
```

Código 1-13

Uso del elemento <nav>.

Tal como puede ver en el Código 1-13, el elemento <nav> también se encuentra entre las etiquetas <body> pero después de la etiqueta de cierre </header>. Se debe a que <nav> no es parte de la cabecera sino de una nueva sección.

Ya hemos mencionado que la estructura y el orden a utilizar con HTML5 es algo muy personal porque HTML5 es muy versátil y solo proporciona los parámetros y elementos básicos para trabajar, mientras que la forma de usarlos es una decisión personal. Gracias a esta versatilidad, la etiqueta <nav> realmente podría ser insertada dentro del elemento <header> o en cualquier otra sección del cuerpo. Sin embargo, debe tener en cuenta que se trata de una etiqueta que ha sido creada para proporcionar más información a los navegadores y para ayudar a los nuevos programas y dispositivos a identificar las partes más relevantes del documento y, como ya hemos dicho antes, para mantener la portabilidad y la facilidad de lectura de nuestro código HTML es preferible seguir las normas. El elemento <nav> está destinado a contener ayudas a la navegación como el menú principal u otros grandes bloques de navegación, y con este objetivo debería ser utilizado.

Conceptos básicos

El elemento <ul>, introducido en el Código 1-13, define una lista de elementos sin un orden específico. Para declarar cada elemento de la lista es necesario usar los elementos <li>, que se anidan dentro del elemento en <ul> (se introducen

entre las etiquetas <ul> de apertura y cierre) y se muestran en la pantalla en el orden en el que son declarados. Para obtener más información acerca de estos elementos básicos de HTML, vaya a nuestro sitio web y siga los enlaces de este capítulo.

1.4.4 <section>

Las siguientes partes del diseño son las que en la Figura 1-1 hemos llamado Información principal y Barra lateral. Como ya hemos mencionado, la Información principal comprende los contenidos más relevantes del documento y puede tener diversas presentaciones. Debido a que el propósito de estas columnas y bloques es más general, el elemento HTML5 que identifica estas secciones se llama simplemente <section>.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <meta name="description" content="Éste es un ejemplo de HTML5">
  <meta name="keywords" content="HTML5, CSS3, JavaScript">
  <title>Éste es el título del documento</title>
  <link rel="stylesheet" href="misestilos.css">
</head>
<body>
  <header>
    <h1>Título principal de la página</h1>
  </header>
  <nav>
    <ul>
      <li>Inicio</li>
      <li>Imágenes</li>
      <li>Vídeos</li>
      <li>Contacto</li>
    </ul>
  </nav>
  <section>

  </section>
</body>
</html>
```

Código 1-14

Uso del elemento <section>.

Igual que sucedía con la Barra de navegación, la Información principal constituye una sección aparte. Por lo tanto, las instrucciones correspondientes a la Información principal están por debajo de la etiqueta </nav> de cierre.

Importante

Las etiquetas que representan cada sección del documento se encuentran en el código en forma de lista, una tras otra, pero en el sitio web de algunas de estas secciones estarán ubicadas unas junto a otras (la información principal y las barras laterales, por ejemplo). En HTML5, la presentación de estos elementos en la pantalla ha sido delegada a CSS y se logra mediante la asignación de estilos CSS para cada elemento. Recuerde que estudiaremos CSS en el próximo capítulo.

Hágalo usted mismo

Compare el contenido del Código 1-14 y el diseño de la Figura 1-3 para comprender cómo se ubican las etiquetas en el código y qué secciones generan esas etiquetas en la representación visual de la página web.

1.4.5 <aside>

En el diseño de la Figura 1-1, la barra lateral está junto a la Información principal. Las etiquetas <aside> definen una columna o sección que normalmente contiene los datos relativos a la información principal, pero no es tan relevante ni importante como aquella. En el ejemplo de un diseño de blog estándar presentado en la Figura 1-2, la Barra lateral contiene una lista de enlaces a cada entrada del blog o proporcionan información adicional sobre el autor del blog. La información que contiene esta sección está por lo tanto relacionada con la Información principal, pero no es relevante por sí misma. Siguiendo el ejemplo del blog, podemos decir que las entradas del blog son relevantes, pero los vínculos y las previsualizaciones cortas de las entradas solo facilitan la navegación y aquello en lo que el usuario o lector estará más interesado.

En HTML5 podemos identificar este tipo de información secundaria con el elemento <aside>.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <meta name="description" content="Éste es un ejemplo de HTML5">
  <meta name="keywords" content="HTML5, CSS3, JavaScript">
  <title>Éste es el título del documento</title>
  <link rel="stylesheet" href="misestilos.css">
</head>
<body>
  <header>
    <h1>Título principal de la página</h1>
  </header>
  <nav>
    <ul>
      <li>Inicio</li>
      <li>Imágenes</li>
      <li>Vídeos</li>
      <li>Contacto</li>
    </ul>
  </nav>
  <section>

  </section>
  <aside>
    <blockquote>Cita del Artículo 1</blockquote>
    <blockquote>Cita del Artículo 2</blockquote>
  </aside>
</body>
</html>
```

Código 1-15

Uso del elemento <aside>.

El elemento <aside> no tiene una posición predefinida, de manera que puede situarse tanto del lado derecho como del izquierdo de la página. Este elemento describe la información que contiene y no su ubicación en la estructura y puede por tanto estar situado en cualquier parte, siempre y cuando su contenido no sea considerado parte de la información principal del documento. Por ejemplo, podemos utilizar el elemento <aside> dentro de un elemento <section>, o incluso dentro de la información principal, por decir algo, para mostrar una cita de texto dentro de la página.

Conceptos básicos

El elemento <blockquote> utilizado para mostrar información relacionada con cada artículo de la página en el Código 1-15 define un bloque de texto que se cita desde otra parte del documento. Los navegadores muestran de este elemento por defecto con los márgenes, proporcionando un formato específico al párrafo que contienen.

1.4.6 <footer>

Para terminar la construcción de la estructura principal de nuestro documento HTML5, solo necesitamos un elemento más. Ya tenemos el encabezado del cuerpo, las secciones con ayudas a la navegación y la información importante, así como información adicional en una barra lateral. Lo único que queda es cerrar el diseño y dar fin al cuerpo del documento. HTML5 proporciona un elemento específico para este propósito llamado <footer>.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <meta name="description" content="Éste es un ejemplo de HTML5">
  <meta name="keywords" content="HTML5, CSS3, JavaScript">
  <title>Éste es el título del documento</title>
  <link rel="stylesheet" href="misestilos.css">
</head>
<body>
  <header>
    <h1>Título principal de la página</h1>
  </header>
  <nav>
    <ul>
      <li>Inicio</li>
      <li>Imágenes</li>
      <li>Vídeos</li>
      <li>Contacto</li>
    </ul>
  </nav>
  <section>

  </section>
  <aside>
    <blockquote>Cita del Artículo 1</blockquote>
    <blockquote>Cita del Artículo 2</blockquote>
  </aside>
  <footer>
    Copyright &copy; 2013
  </footer>
</body>
</html>
```

Código 1-16

Uso del elemento <footer>.

En el diseño de una página web estándar (Figura 1-1), la sección denominada la Barra institucional se definiría con etiquetas <footer>. Estas etiquetas identifican normalmente el final (o el pie) del documento, que es la parte de la página web habitualmente utilizada para compartir información general sobre el autor o la empresa detrás del proyecto, al igual que los derechos de autor, términos y condiciones, etc.

Tenga en mente que, aunque por lo general el elemento <footer> representa el final del cuerpo del documento y tiene el propósito principal descrito, también puede ser utilizado varias veces dentro del cuerpo para representar también el final de diferentes secciones. (así como también las etiquetas <header> pueden ser utilizadas varias veces en el cuerpo.) Estudiaremos este uso más adelante.

1.5 En el interior del cuerpo

El cuerpo de nuestro documento está listo, pero, aunque la estructura básica del sitio está terminada, aún tenemos que desarrollar el contenido. Los elementos HTML5 estudiados hasta ahora nos ayudan a identificar cada segmento del diseño y la finalidad intrínseca de cada uno, pero lo realmente importante para nuestro sitio web es el contenido de cada uno de esos segmentos.

La mayor parte de los elementos ya analizados fueron creadas para proporcionar una estructura para el documento HTML que pueda ser identificada y reconocida por los navegadores y dispositivos. Hemos conocido las etiquetas <body>, que declaran el cuerpo o la parte visible del documento, las etiquetas <header>, que encierran información importante del cuerpo, las etiquetas <nav>, que suministran ayudas a la navegación, las etiquetas <section>, que incluyen el contenido más relevante, y las etiquetas <aside> y <footer>, que proporcionan información adicional.

Sin embargo, ninguno de estos elementos declara nada sobre el contenido propiamente, sino que tienen una función estructural muy específica.

A medida que avanzamos en la creación del documento HTML, nos acercamos a la definición de los contenidos. Esta información estará compuesta de diferentes elementos visuales tales como títulos, textos, imágenes, vídeos y aplicaciones interactivas, entre otros. Es necesario que sea posible diferenciar estos elementos y establecer relaciones entre ellos.

1.5.1 <article>

El diseño básico con el que venimos trabajando (Figura 1-1) es hoy en día la estructura más común y básica para sitios web en Internet, pero también ilustra cómo son mostrados en pantalla los contenidos más relevantes.

De la misma manera que los blogs se dividen en entradas, los sitios web suelen presentar la información relevante dividida en partes que comparten características similares. El elemento <article> es el que nos permite identificar cada una de estas partes.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <meta name="description" content="Éste es un ejemplo de HTML5">
  <meta name="keywords" content="HTML5, CSS3, JavaScript">
  <title>Éste es el título del documento</title>
  <link rel="stylesheet" href="misestilos.css">
</head>
<body>
  <header>
    <h1>Título principal de la página</h1>
  </header>
  <nav>
    <ul>
      <li>Inicio</li>
      <li>Imágenes</li>
      <li>Vídeos</li>
      <li>Contacto</li>
    </ul>
  </nav>
  <section>
    <article>
      Éste es el texto de mi primera entrada
    </article>
    <article>
      Éste es el texto de mi segunda entrada
    </article>
  </section>
  <aside>
    <blockquote>Cita del Artículo 1</blockquote>
    <blockquote>Cita del Artículo 2</blockquote>
  </aside>
  <footer>
    Copyright &copy; 2013
  </footer>
</body>
</html>
```

Código 1-17

Uso del elemento <article>.

Como se puede ver en el Código 1-17 que acabamos de presentar, las etiquetas <article> se encuentran dentro de las etiquetas <section>. Las etiquetas <article> están subordinadas a la sección <section> de la misma manera que cualquier elemento colocado entre las etiquetas del elemento <body> está subordinado a este último. E igual que sucede con cualquier elemento anidado dentro del elemento <body>, las etiquetas <article> se colocan una tras otra, y cada una constituye una parte independiente del elemento <section>, como se muestra en la Figura 1-4.

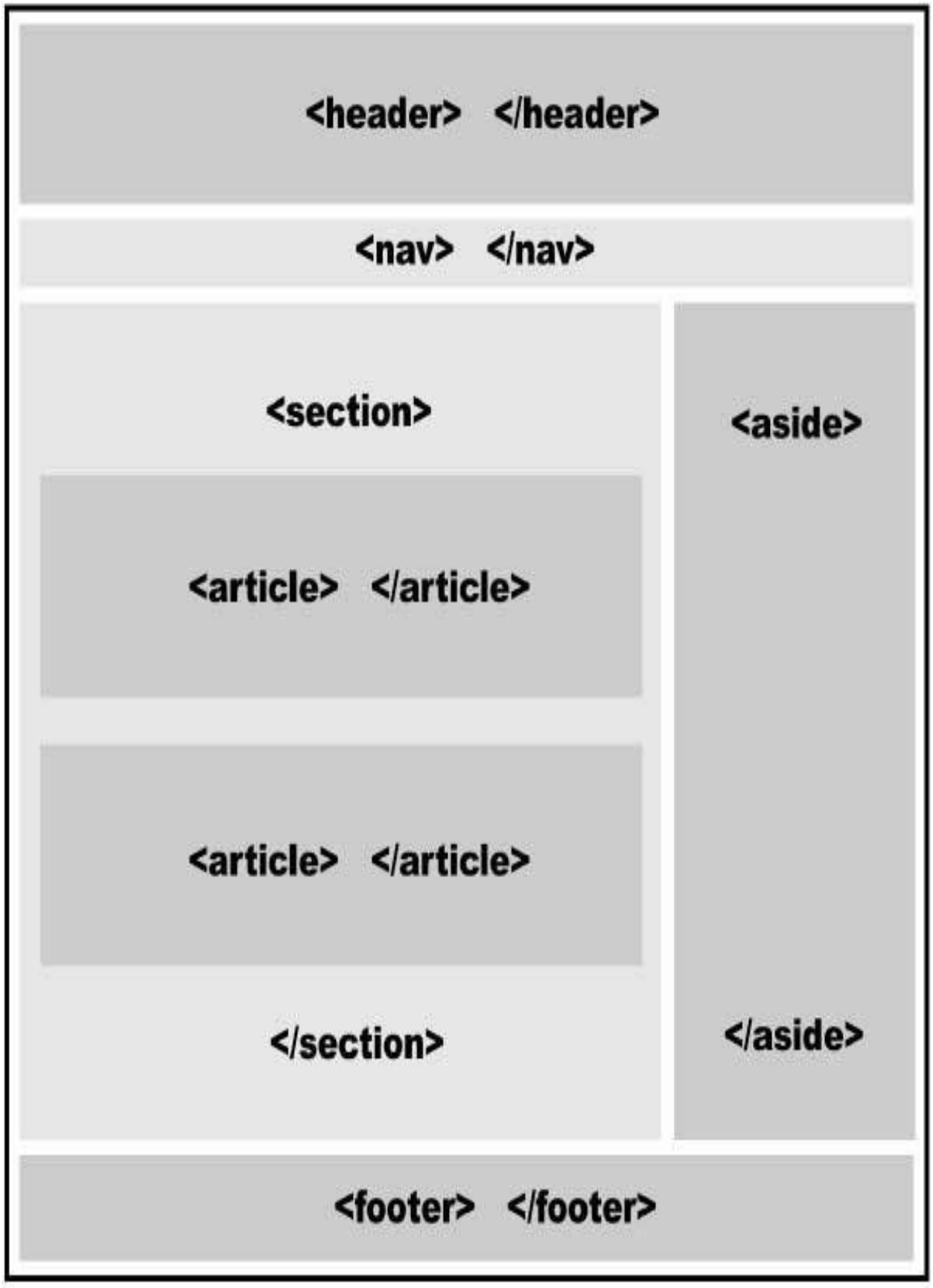

<header> </header>
<nav> </nav>
<section>
<aside>
<article> </article>
<article> </article>
</section>
</aside>
<footer> </footer>

Figura 1-4

Representación visual de las etiquetas <article> dentro de la sección creada para albergar la información relevante de la página web.

Conceptos básicos

Ya hemos explicado antes que la estructura HTML puede ser descrita como un árbol cuya raíz es el elemento <html>. Otra forma de describir las relaciones entre los elementos es nombrarlos como padres, hijos o hermanos de acuerdo a su posición en la estructura de árbol. Por ejemplo, en un documento HTML típico, el elemento <body> es hijo del elemento <html> y hermano del elemento <head>. Ambos, <body> y <head>, tienen al elemento <html> como padre.

El elemento <article> no está limitado a artículos de noticias, por ejemplo, sino que está destinado a contener cualquier elemento de contenido independiente. Por tanto, puede incluir ya sea una entrada de un foro, un artículo de una revista, una entrada de un blog, un comentario de usuario, o cualquier otro contenido similar. Este elemento, <article>, agrupará segmentos de información relacionados entre sí, independientemente de la naturaleza de la información.

Como una parte independiente del documento, el contenido de cada elemento <article> tendrá su propia estructura independiente. Para definir esta estructura, podemos tomar ventaja de la versatilidad de las etiquetas <header> y <footer> estudiadas anteriormente. Estas etiquetas son portátiles y pueden ser utilizadas no solo en el cuerpo sino también en cada sección de nuestro documento.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <meta name="description" content="Éste es un ejemplo de HTML5">
  <meta name="keywords" content="HTML5, CSS3, JavaScript">
  <title>Éste es el título del documento</title>
  <link rel="stylesheet" href="misestilos.css">
</head>
<body>
  <header>
    <h1>Título principal de la página</h1>
  </header>
  <nav>
    <ul>
      <li>Inicio</li>
      <li>Imágenes</li>
      <li>Vídeos</li>
      <li>Contacto</li>
    </ul>
   </nav>
  <section>
    <article>
      <header>
        <h1>Título de la primera entrada</h1>
      </header>
      Éste es el texto de la primera entrada.
      <footer>
        <p>Comentarios (0)</p>
      </footer>
    </article>
    <article>
      <header>
        <h1>Título de la segunda entrada</h1>
      </header>
      Texto de la segunda entrada
      <footer>
        <p>Comentarios (0)</p>
      </footer>
    </article>
  </section>
  <aside>
    <blockquote>Cita del Artículo 1</blockquote>
    <blockquote>Cita del Artículo 2</blockquote>
  </aside>
  <footer>
    Copyright &copy; 2013
  </footer>
</body>
</html>
```

Código 1-18

Construcción de la estructura <article>.

Conceptos básicos

El elemento <p> que hemos usado dentro del pie o, lo que es lo mismo, dentro del elemento <footer> de cada artículo del Código 1-18, define un párrafo. Por defecto, los navegadores muestran los contenidos de estas etiquetas con márgenes y saltos de línea. Además de declarar párrafos, también es posible sacar provecho de las propiedades de este elemento para aplicar formato a textos breves, que es el uso que se le ha dado en este ejemplo.

Los dos artículos insertados en el Código 1-18 se construyeron con el elemento <article> y tienen una estructura específica. En primer lugar, las etiquetas <header> contienen el título, que se define con un elemento <h1>. A continuación está el contenido en sí, que es el texto del artículo. Y finalmente, después del texto, está la etiqueta <footer> que especifica en este caso el número de comentarios.

1.5.2 <hgroup>

Dentro de cada elemento <header>, ya sea al comienzo del cuerpo o al comienzo de cada elemento <article> de nuestro ejemplo, hemos incorporado etiquetas <h1> para indicar un título. Las etiquetas <h1> son, básicamente, todo lo que se necesita para crear el encabezamiento de todas las partes del documento, pero a veces también hay que añadir subtítulos o más información para declarar de qué trata la página web o alguna de sus secciones. De hecho, es habitual que el elemento <header> albergue también otros elementos, como tablas de contenidos, formas de búsqueda, o textos cortos y logotipos.

Puede hacer uso de todas las etiquetas H: <h1>, <h2>, <h3>, <h4>, <h5> y <h6>. Sin embargo, para fines de procesamiento interno y para evitar la generación de múltiples secciones o subsecciones durante la interpretación del documento, estas etiquetas deben estar agrupadas. Para ello, HTML5 proporciona el elemento <hgroup>.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <meta name="description" content="Éste es un ejemplo de HTML5">
  <meta name="keywords" content="HTML5, CSS3, JavaScript">
  <title>Éste es el título del documento</title>
  <link rel="stylesheet" href="misestilos.css">
</head>
<body>
  <header>
    <h1>Título principal de la página</h1>
  </header>
  <nav>
    <ul>
      <li>Inicio</li>
      <li>Imágenes</li>
      <li>Vídeos</li>
      <li>Contacto</li>
    </ul>
   </nav>
  <section>
    <article>
      <header>
        <hgroup>
          <h1>Título de la primera entrada</h1>
          <h2>Subtítulo de la primera entrada</h2>
        </hgroup>
        <p>Publicada el 12-01-2013</p>
      </header>
      Texto de la primera entrada
      <footer>
        <p>Comentarios (0)</p>
      </footer>
    </article>
    <article>
      <header>
        <hgroup>
          <h1>Título de la segunda entrada</h1>
          <h2>Subtítulo de la segunda entrada</h2>
        </hgroup>
        <p>Publicada el 15-01-2013</p>
      </header>
      Texto de la segunda entrada
      <footer>
        <p>Comentarios (0)</p>
      </footer>
    </article>
  </section>
   <aside>
    <blockquote>Cita del Artículo 1</blockquote>
    <blockquote>Cita del Artículo 2</blockquote>
  </aside>
  <footer>
    Copyright &copy; 2013
  </footer>
</body>
</html>
```

Código 1-19

Uso del elemento <hgroup>.

Las etiquetas H deben respetar su jerarquía, es decir, primero se debe declarar el título con la etiqueta <h1>, a continuación, utilizar <h2> para los subtítulos, y así sucesivamente. Sin embargo, a diferencia de las versiones anteriores de HTML, HTML5 permite reutilizar las etiquetas H y construir esta jerarquía una y otra vez en cada sección del documento. En el ejemplo del Código 1-19, se añade un subtítulo y metadatos para cada entrada y agrupamos a títulos y subtítulos con la etiqueta <hgroup>. La jerarquía <h1> y <h2> es reutilizada en cada elemento <article>.

Importante

El elemento <hgroup> es necesario cuando hay un título y un subtítulo, o más etiquetas H juntas en el mismo elemento <header>. Este elemento solo puede contener etiquetas H y por eso en el ejemplo lo mantuvimos fuera de los metadatos. Si solo dispone de la etiqueta <h1> o la etiqueta <h1> y metadatos, no tiene que agrupar estos elementos. Por ejemplo, en el elemento <header> del cuerpo no las usamos porque éste solo contenía un elemento H dentro. Recuerde siempre que las etiquetas <hgroup> han sido diseñadas para agrupar solamente las etiquetas H, tal como como su nombre indica.

Los navegadores y programas que ejecutan sitios web leen el código HTML y crean su propia estructura interna para interpretar y procesar cada elemento. Esta

estructura interna se divide en secciones que son independientes de las divisiones en el diseño o del elemento <section>. Estas son las secciones conceptuales generadas durante la interpretación del código. El elemento <header> no crea una de estas secciones conceptuales por sí mismo, lo que significa que los elementos dentro del <header> representan diferentes niveles y podrían generar internamente secciones diferentes. El elemento <hgroup> fue creado con el propósito de agrupar etiquetas H y evitar errores de interpretación por parte del navegador.

1.5.3 <figure> y <figcaption>

La etiqueta <figure> fue creada para ser declarar el contenido del documento de una forma más específica. Antes de que fuera introducido este elemento, no era posible identificar el contenido que formaba parte de la información, sino el que era independiente, por ejemplo, ilustraciones, fotos, videos, etc. Por lo general, esos elementos forman parte de los contenidos relevantes pero pueden ser removidos sin afectar o interrumpir el flujo de un documento. Cuando este tipo de información está presente, puede ser identificada con las etiquetas <figure>.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <meta name="description" content="Éste es un ejemplo de HTML5">
  <meta name="keywords" content="HTML5, CSS3, JavaScript">
  <title>Éste es el título del documento</title>
  <link rel="stylesheet" href="misestilos.css">
</head>
<body>
  <header>
    <h1>Título principal de la página</h1>
  </header>
  <nav>
    <ul>
      <li>Inicio</li>
      <li>Imágenes</li>
      <li>Vídeos</li>
      <li>Contacto</li>
    </ul>
   </nav>
  <section>
    <article>
      <header>
        <hgroup>
          <h1>Título de la primera entrada</h1>
          <h2>Subtítulo de la primera entrada</h2>
        </hgroup>
        <p>Publicada el 12-01-2013</p>
      </header>
      Texto de la primera entrada
      <figure>
        <img src="http://minkbooks.com/content/myimage.jpg">
        <figcaption>
           Imagen de la primera entrada
         </figcaption>
      </figure>
       <footer>
        <p>Comentarios (0)</p>
      </footer>
    </article>
    <article>
      <header>
        <hgroup>
          <h1>Título de la segunda entrada</h1>
          <h2>Subtítulo de la segunda entrada</h2>
        </hgroup>
        <p>Publicada el 15-01-2013</p>
      </header>
      Texto de la segunda entrada
      <footer>
        <p>Comentarios (0)</p>
      </footer>
    </article>
  </section>
   <aside>
    <blockquote>Cita del Artículo 1</blockquote>
    <blockquote>Cita del Artículo 2</blockquote>
  </aside>
  <footer>
    Copyright &copy; 2013
  </footer>
</body>
</html>
```

Código 1-20

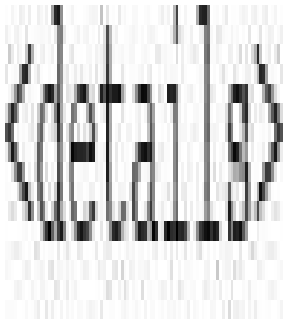
Uso de los elementos <figure> y <figcaption>.

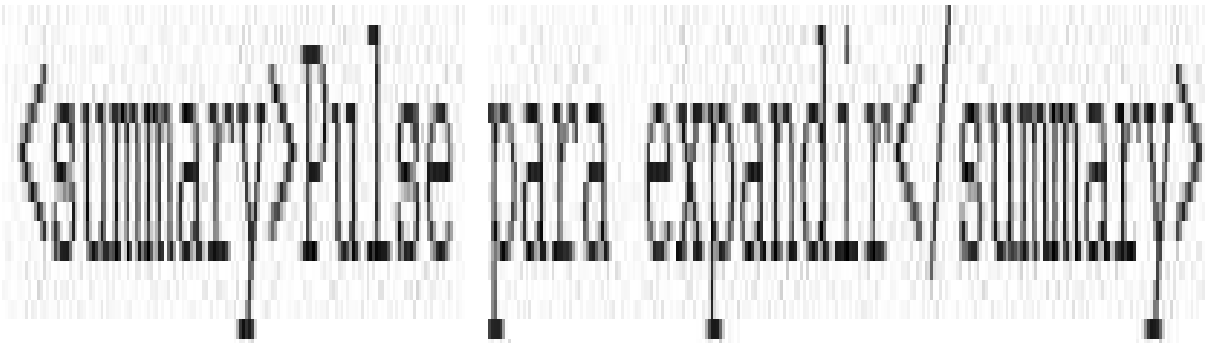
En el Código 1-20, insertamos en la primera entrada una imagen (<img src="http://www.minkbooks.com/content/myimage.jpg">) después del texto del mensaje. Ésta es una práctica habitual, pues a menudo el texto está enriquecido con imágenes o videos. Las etiquetas <figure> nos permiten encerrar estos elementos visuales y diferenciarlos de la información más relevante.

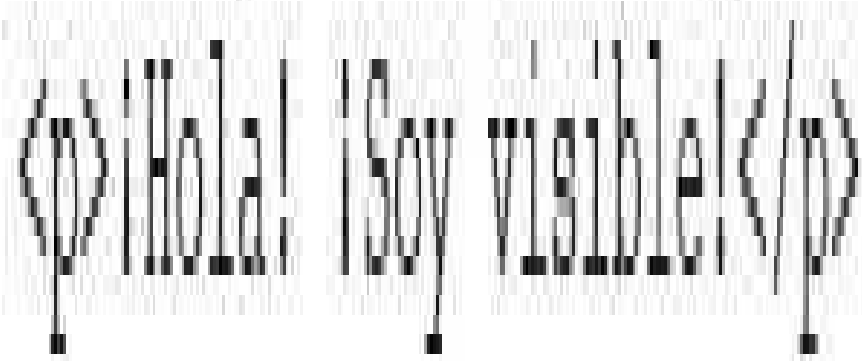
También en el Código 1-20 se encuentra un elemento adicional dentro del elemento <figure>. Por lo general, las unidades de información como imágenes o vídeos vienen acompañadas de un breve texto, normalmente ubicado bajo ellas, que las describe. HTML5 aporta un elemento para ubicar e identificar esta leyenda descriptiva. Las etiquetas <figcaption> encierran la leyenda relacionada con el elemento <figure> y establecen una relación entre ambos elementos y su contenido.

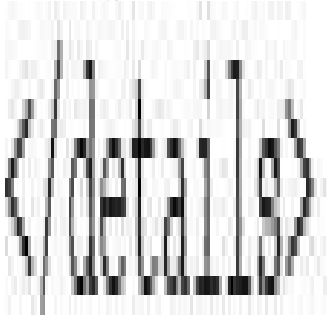
1.5.4 <details> y <summary>

Una característica importante de los sitios web es la posibilidad de mostrar información adicional cuando así es solicitado por el usuario. Para evitar el uso de Javascript y facilitar la creación de esta herramienta, HTML5 incorpora los elementos <details> y <summary>. El elemento <details> declara la herramienta de ampliación de información y, dentro de este elemento, se especifica el título de la herramienta con el elemento <summary> además de toda la información que ésta oculta.


<details>

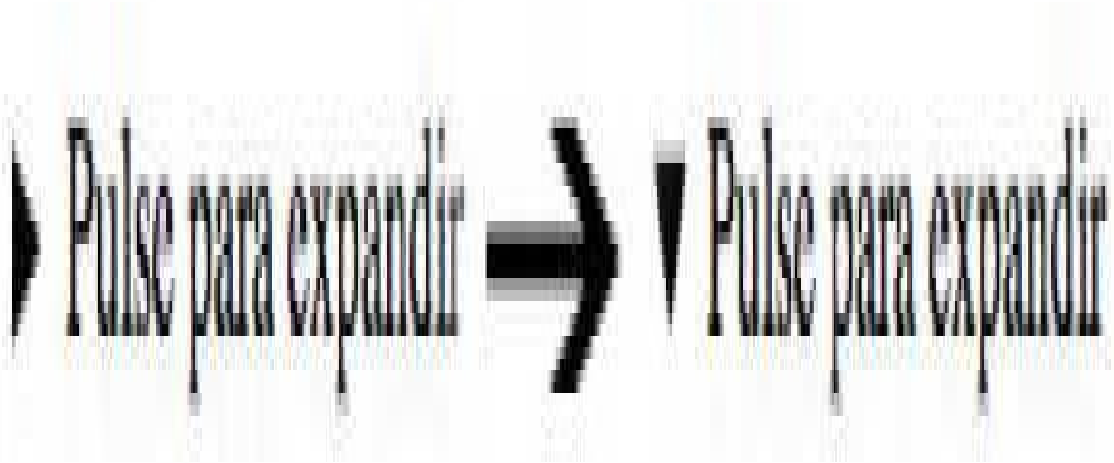

<summary>Pulse para expandir</summary>


¡Hola! ¡Soy visible!


</details>

Código 1-21

Creación de una herramienta con <details> y <summary>.



¡Hola! ¡Soy visible!

Figura 1-15

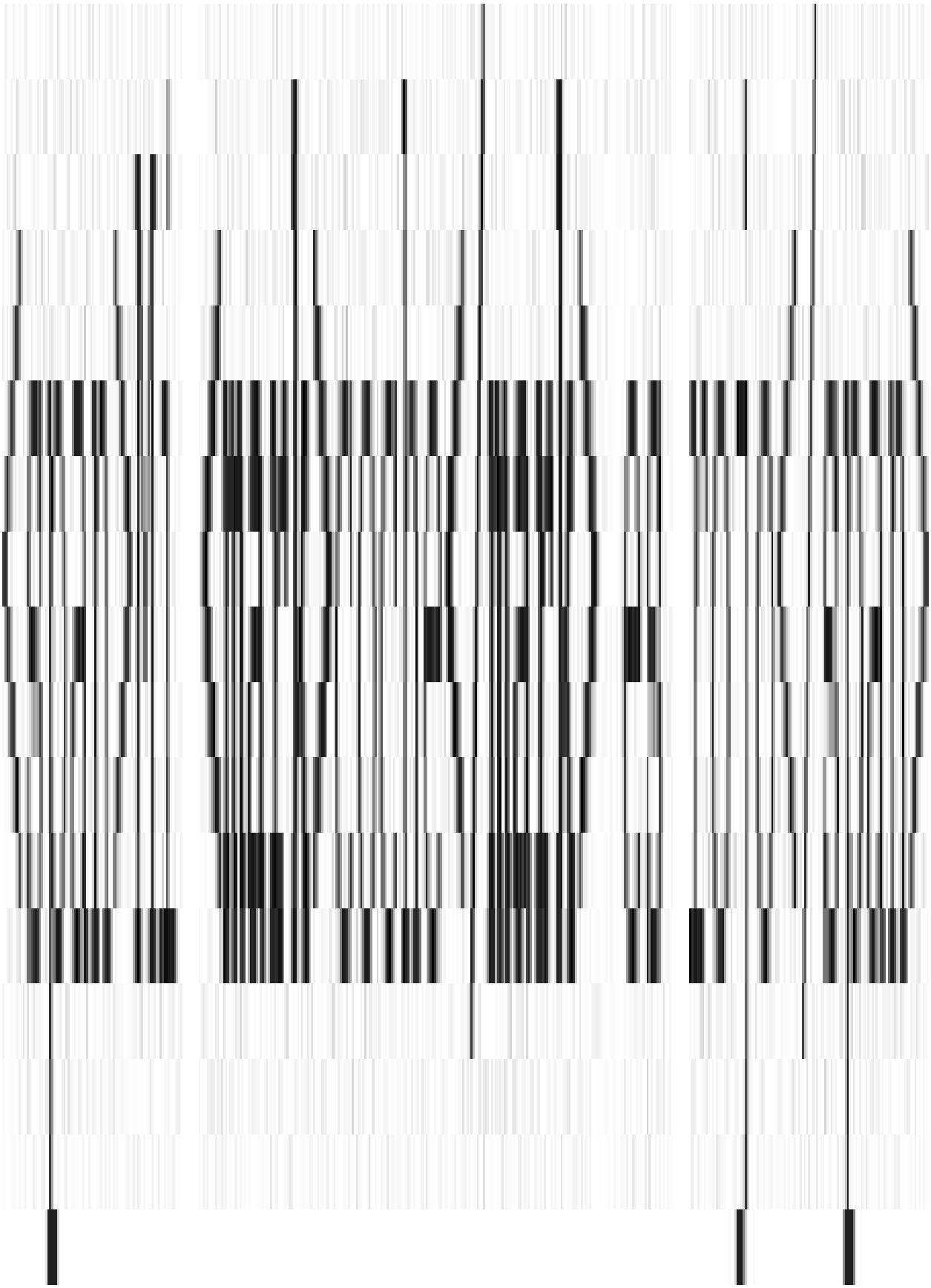
El elemento <details> antes y después de pulsar sobre él.

1.6 Elementos nuevos y elementos antiguos

HTML5 ha sido desarrollado con el fin de simplificar, precisar y organizar el código. Para lograr estos propósitos, se añadieron elementos y atributos y se integró HTML con CSS y Javascript. Estas incorporaciones y mejoras respecto a las versiones anteriores no se refieren solo a los nuevos elementos sino también a la forma en la que usamos los viejos.

1.6.1 <mark>

La etiqueta <mark> se añadió para resaltar parte de un texto que en principio no es considerada importante, pero que puede adquirir relevancia de acuerdo con la actividad del usuario. El mejor ejemplo es un resultado de búsqueda. El elemento <mark> resaltar la parte del texto que coincide con la cadena de búsqueda.



Código 1-22

Uso del elemento <mark> para resaltar la palabra “coche”.

Si alguien realiza una búsqueda en una página web para la palabra “coche”, los resultados podrían ser mostrados con el Código 1-22. El breve texto representa los resultados de la búsqueda, y las etiquetas <mark> encierran el texto que se ha buscado (la palabra “coche” en nuestro ejemplo). En algunos navegadores, la palabra buscada se resalta con un fondo amarillo por defecto, pero siempre es posible sobrescribir esos estilos usando su propio CSS, como veremos en capítulos posteriores.

En el pasado, generalmente se lograba el mismo resultado utilizando un elemento <b>. Sin embargo, la adición de <mark> ayudó a cambiar el sentido y establecer un nuevo propósito para estos y otros elementos relacionados.

: Se utiliza para indicar énfasis (en sustitución de la etiqueta <i> que hemos utilizado antes).

: Es para indicar importancia.

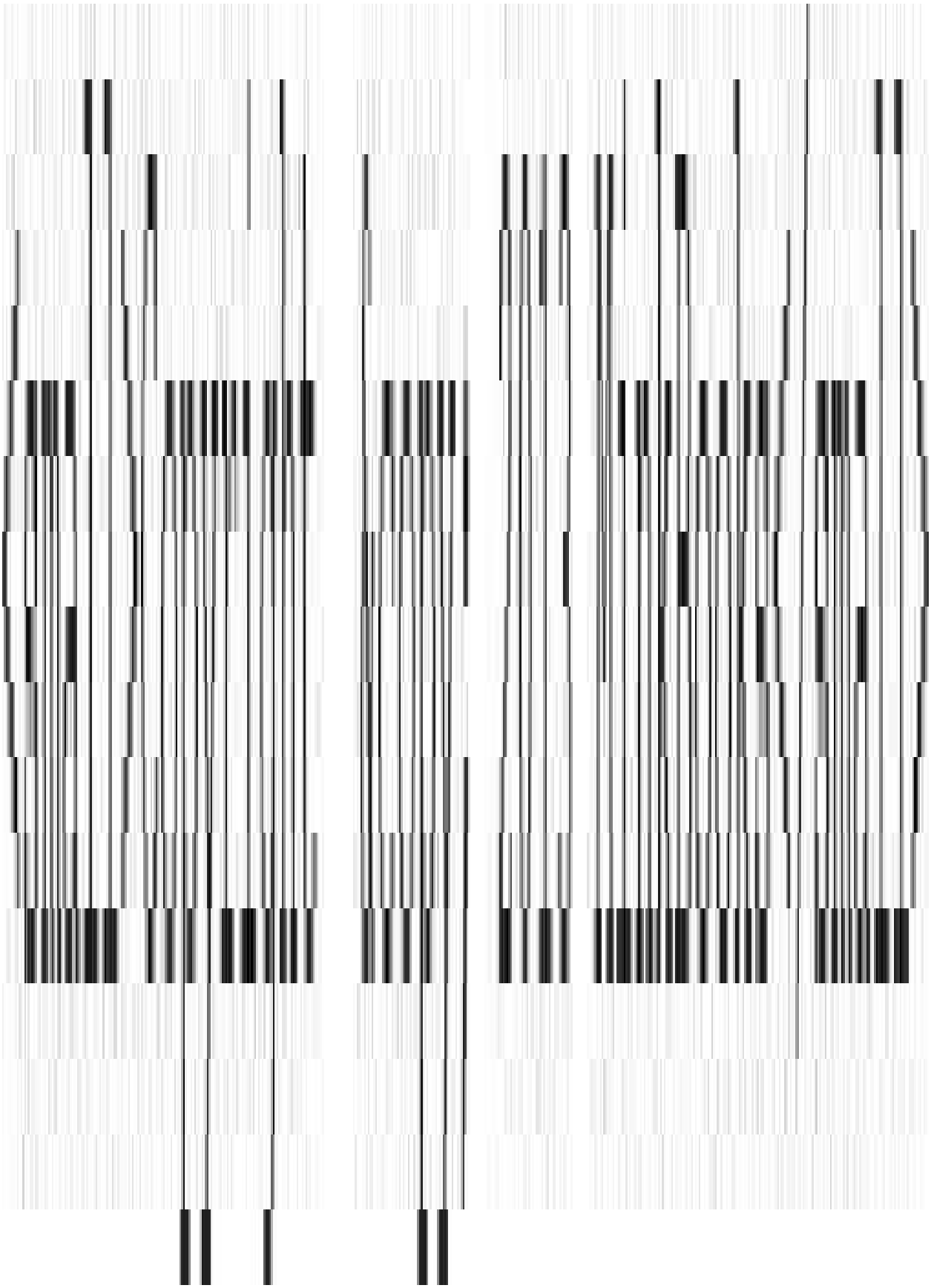
<mark>: Destaca texto que es pertinente de acuerdo con las circunstancias.

<b>: Debe ser utilizado solo cuando no hay ningún otro elemento apropiado para la situación.

<i>: Ya no declara cursiva a un texto, pero de acuerdo a la especificación, representa una voz alternativa o humor, como un pensamiento, un término técnico, etc.

1.6.2 <small>

La nueva especificación de HTML también es evidente en elementos como <small>. Anteriormente, este elemento presentaba cualquier texto en fuente pequeña. La palabra clave hacía referencia al tamaño del texto, independientemente de su significado. En HTML5, el propósito del elemento <small> es representar la “letra pequeña” en sentido figurado, es decir, textos legales, renuncias, etc.

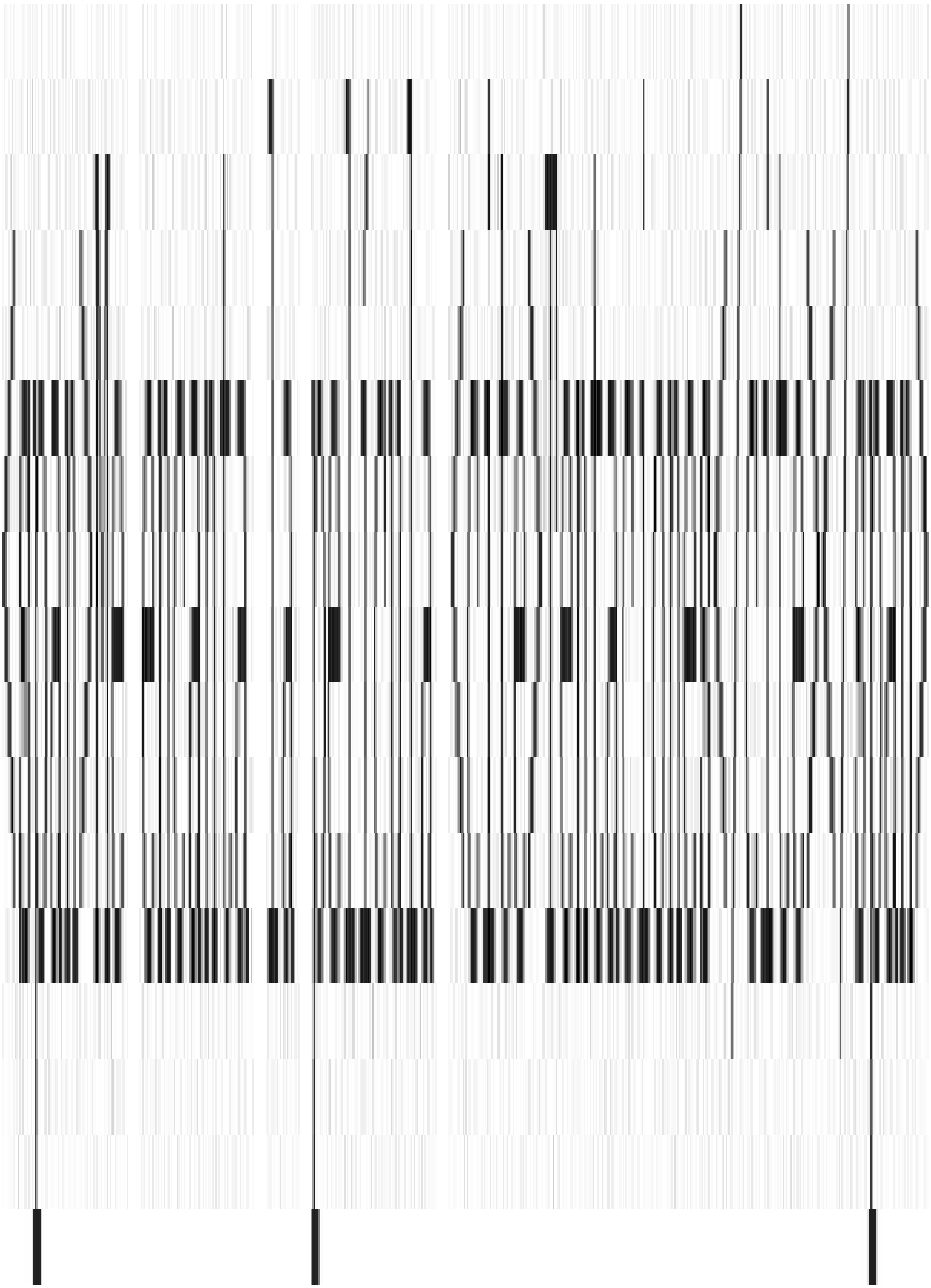


Código 1-23

Texto legal con <small>.

1.6.3 <cite>

Otro elemento que se ha vuelto más específico es <cite>. Ahora estas etiquetas encierran el título de una obra como un libro, una película, una canción, etc.



Código 1-24

Citar una película con <cite>.

1.6.4 <address>

El elemento <address> es un viejo elemento que se ha hecho estructural. No hemos tenido que utilizarlo en la construcción de nuestro documento de ejemplo, pero puede encajar perfectamente en algunas situaciones para representar la información de contacto de un elemento o <article> del elemento <body> entero. Este elemento debe ser incluido dentro de un elemento <footer>, como en el ejemplo siguiente:

```
<article>
  <header>
    <h1>Título del segundo post</h1>
  </header>
  Éste es el texto del artículo
  <footer>
    <address>
      <a href="http://www.jdgauchat.com">J.D. Gauchat</a>
    </address>
  </footer>
</article>
```

Código 1-25

Agregar información de contacto para un elemento <cite>.

Conceptos básicos

El hipertexto es parte de la naturaleza de HTML5: un texto que presenta referencias a otros textos y permite a los lectores moverse o “navegar” a través de los documentos. El elemento <a>, implementado en el Código 1-25, se utiliza para crear los hipervínculos necesarios para proporcionar esta característica. Como ya se ha mencionado, este elemento tiene un atributo llamado href para declarar la ruta del documento al que estamos apuntando.

1.6.5 <wbr>

La etiqueta <br>, un elemento que no usa etiqueta de cierre, era anteriormente el único elemento utilizado para insertar un salto de línea en un párrafo. Los navegadores muestran un salto de línea en el lugar de este elemento. Pero solo es útil cuando se sabe exactamente dónde debe generarse el salto de línea. HTML5 introduce el nuevo elemento <wbr> que sugiere la posibilidad de un salto de línea si éste es necesario y asegura así que el navegador no rompa las líneas en los lugares equivocados. En el siguiente ejemplo, el navegador generará un salto de línea después de mostrar la primera línea de texto y, a continuación, de acuerdo con el espacio disponible, o mostrará la segunda línea o bien producirá un salto de línea después del segundo dominio:

```
<article>
  <header>
    <h1>Título de la entrada</h1>
  </header>
  Este es el texto del artículo.<br>
  Visite esta URL: www.minkbooks.com<wbr>/content/bricks.jpg
</article>
```

Código 1-26

Implementación de los elementos <br> y <wbr>.

1.6.6 <time>

En el documento del Código 1-20 se incluyó la fecha en cada elemento <article> para indicar cuándo fue publicado. Utilizamos simplemente un elemento <p> dentro del <header> de los artículos para mostrar la fecha, pero debe saber que hay un elemento especial en HTML5 para este propósito específico. El elemento <time> permite declarar una marca de tiempo que es legible tanto para navegadores como para seres humanos, que representa la fecha y la hora.

```
<article>
  <header>
    <h1>Título del segundo post</h1>
    <time datetime="2012-01-12" pubdate>Publicado 12-01-2013</time>
  </header>
  Este es el texto del artículo
</article>
```

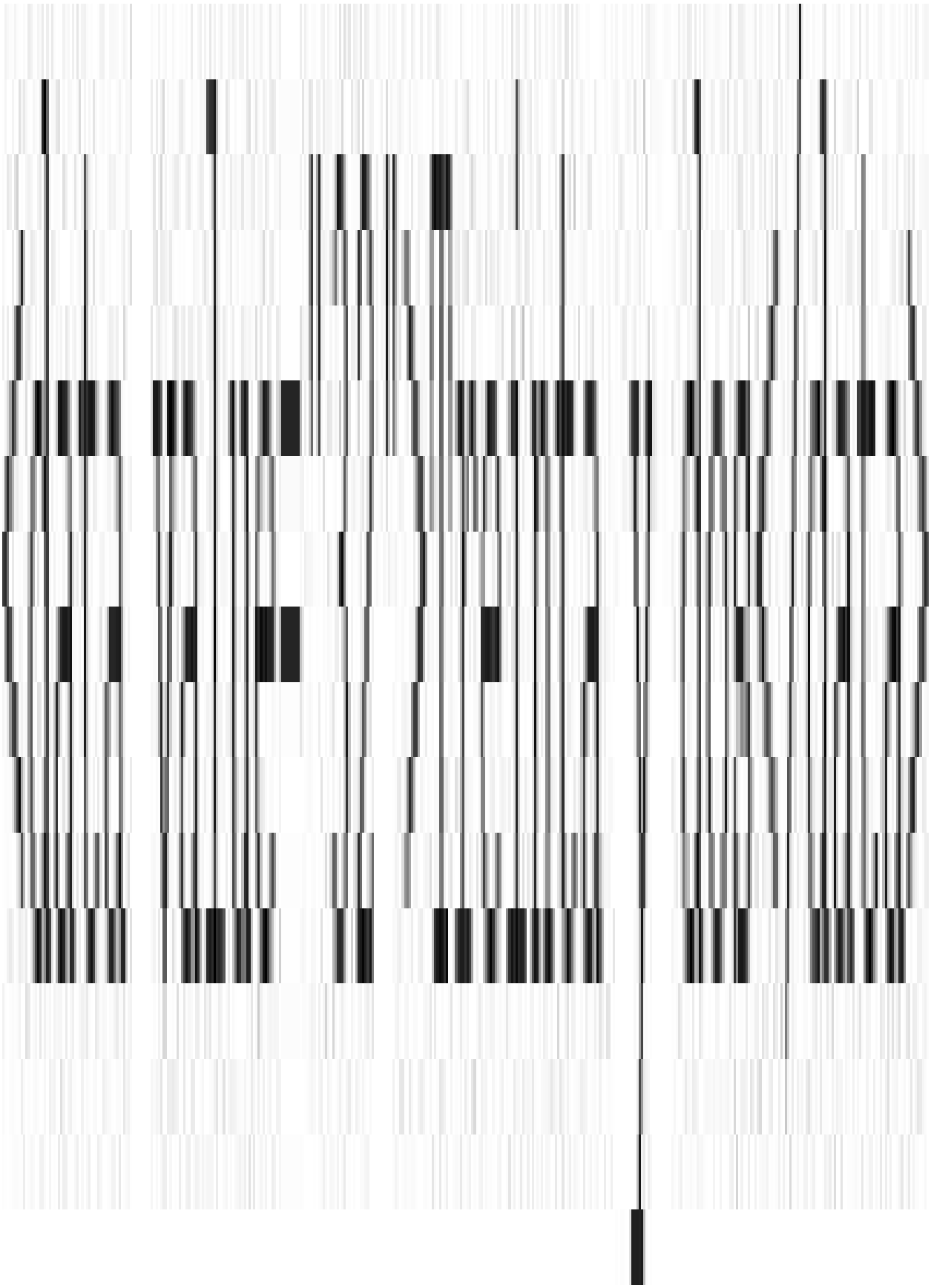
Código 1-27

Fecha y hora utilizando el elemento <hora>.

En el Código 1-27, el elemento <p> utilizado en ejemplos anteriores es sustituido por el nuevo elemento <time> para mostrar la fecha en la que el artículo fue publicado. El atributo datetime tiene un valor que representa la fecha y hora y es legible para un ordenador. El formato de esta marca de tiempo por lo general sigue un patrón, como en este ejemplo: 2013-01-12T12: 10:45. También incluimos el atributo pubdate, que se añade para indicar que el valor del atributo datetime representa la fecha de publicación.

1.6.7 <data>

Para datos que no están relacionados con fecha y hora, se puede utilizar el elemento <data>. Este elemento utiliza el atributo value para representar el valor legible para navegadores.



Codigo 1-28

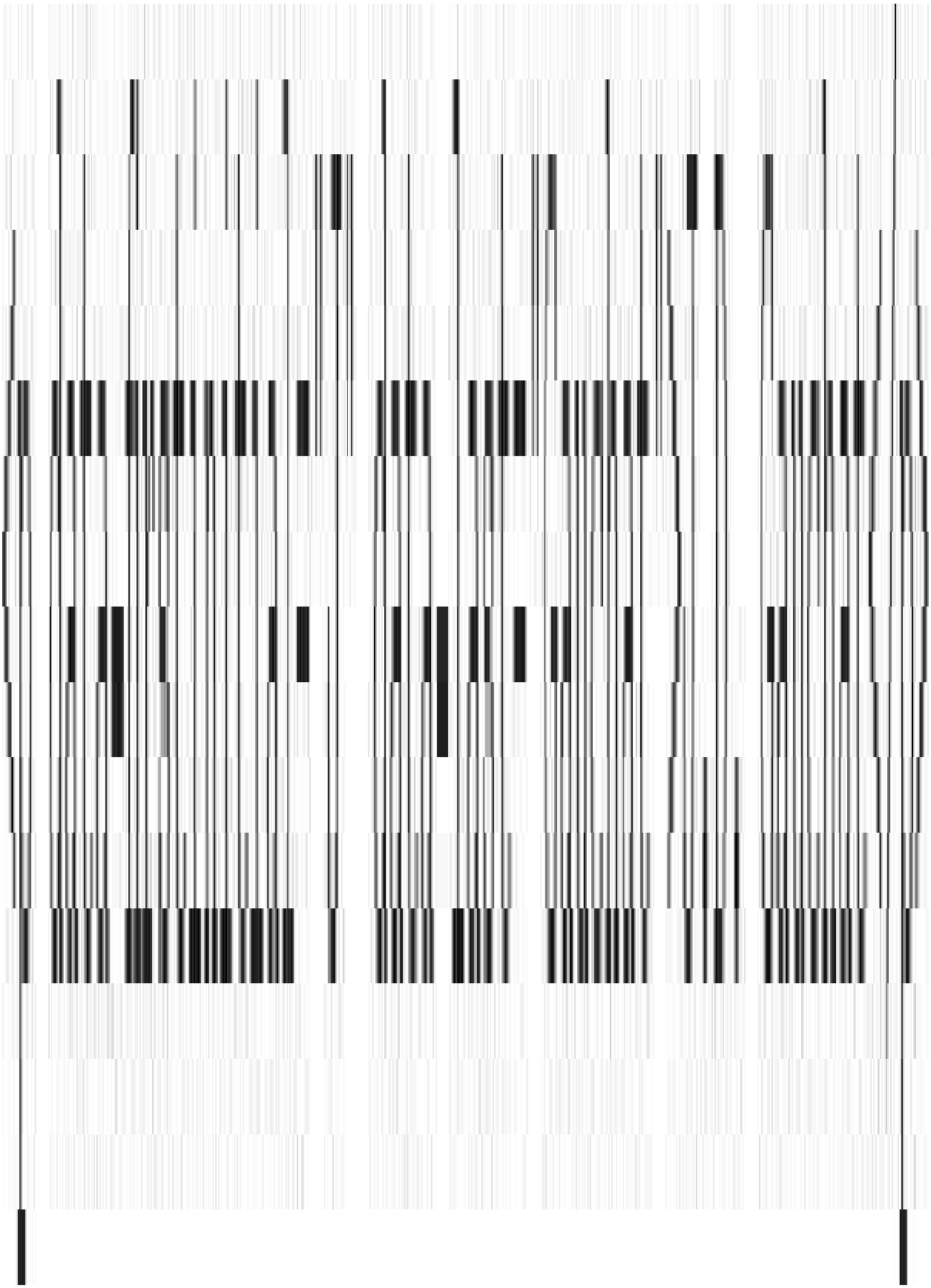
Uso del elemento <data>.

1.7 Nuevos atributos y viejos atributos

En los primeros días de desarrollo de la red, el fin de los atributos era proporcionar estilo a los elementos a los que aquellos eran aplicados. Con la integración de HTML, CSS y Javascript, esto cambió. Ahora los atributos se centran en declarar el tipo de contenido que contiene el elemento y la función que cumplirá. Para lograr esto, HTML5 modificó las características de algunos de los viejos atributos y añadió otros atributos nuevos.

*1.7.1 El atributo data-**

Si necesita declarar datos adicionales, no solo para los elementos <data> y <tiempo> sino para cualquier elemento HTML, entonces el atributo data* es la solución. Este atributo permite asignar información adicional a un elemento sin exponerla al usuario. Se trata de información que está destinada a ser utilizada por la página web y es accesible desde el código Javascript, tal como estudiaremos más adelante. El nombre del atributo está compuesto por el prefijo data- seguido por un nombre personalizado.



Código 1-29

Usar el atributo de data-*.

Importante

Vamos a trabajar con el atributo data-* y la propiedad dataset en el Capítulo 4.

1.7.2 reversed

En el documento HTML del Código 1-20, nos aprovechamos de los tradicionales elementos <ul> y <li> para generar una lista de elementos. En ese caso, el orden de los elementos no era particularmente importante, así que el uso del elemento <ul> era correcto pero, cuando el orden es relevante, se debe utilizar en su lugar el elemento <ol>. Este elemento genera una lista en orden ascendente identificada con los números por defecto. La mejora en HTML5 es la adición del atributo booleano reversed para crear una lista de artículos en orden descendente.

```
<ol reversed>
    <li>Inicio</li>
    <li>Fotos</li>
    <li>Videos</li>
    <li>Contacto</li>
</ol>
```

Código 1-30

Generar una lista en orden descendente.

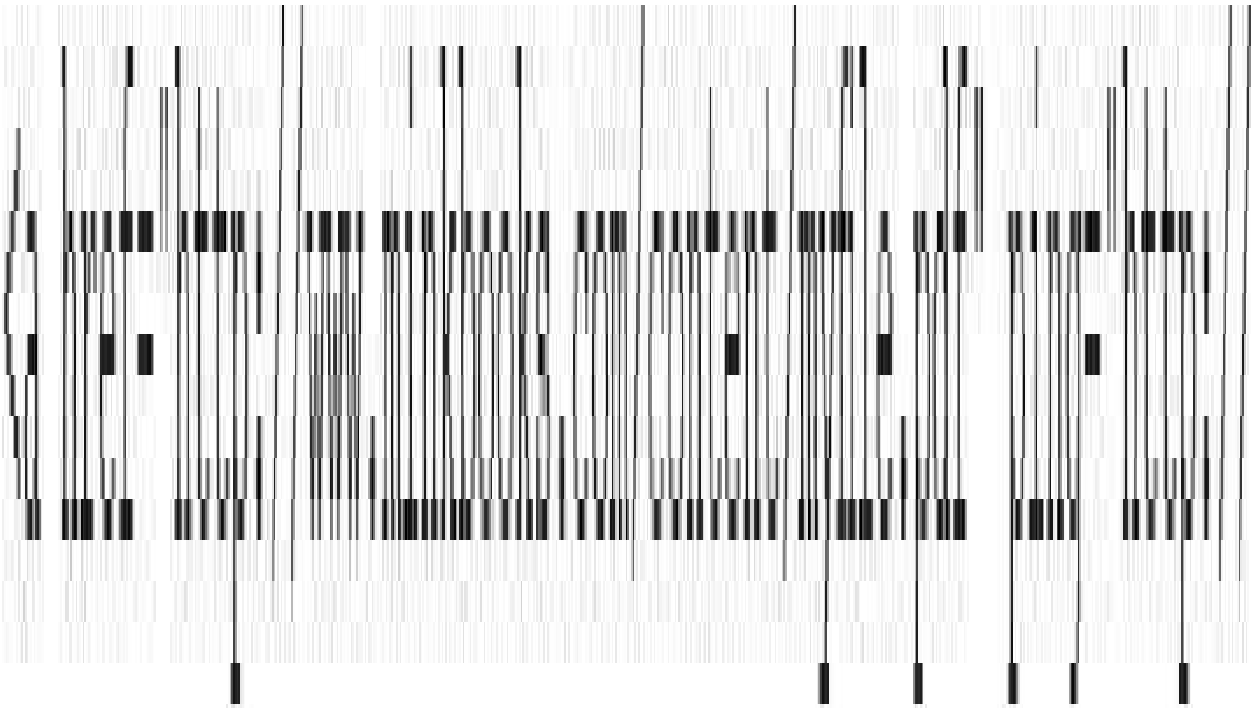
Los elementos siempre se muestran en el orden en que fueron declarados, pero los números que hacen referencia a ellos reflejan el orden ascendente o descendente.

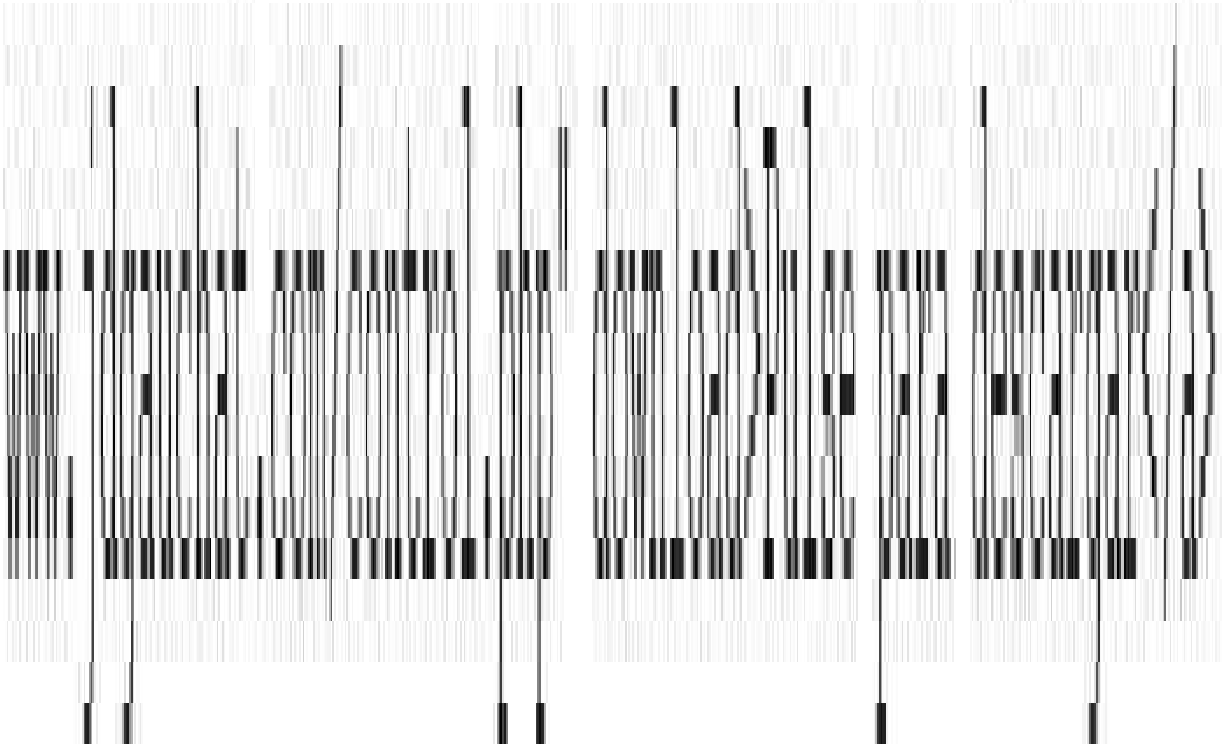
1.7.3 ping y download

El elemento <a> es probablemente el elemento más importante del HTML. Define la característica más importante del lenguaje: el hipervínculo. Fue creado para vincular un documento a otro, permitiendo a los usuarios navegar y acceder fácilmente a la información proporcionada por el sitio web. Pero ahora hacemos mucho más que acceder a documentos. Los usuarios necesitan ser capaces de descargar los documentos y los desarrolladores necesitan poder realizar seguimiento de nuestra actividad. HTML5 incorpora dos nuevos atributos que conservan la vigencia de este elemento y añadir la funcionalidad requerida:

ping: Este atributo declara la ruta de la URL e informa cuando el usuario hace clic en el enlace. El valor puede ser una o más URL separadas por un espacio.

download: Éste es un atributo booleano que indica al navegador que debe descargar el archivo en lugar de leerlo.





Código 1-31

Aplicación de los atributos ping y download.

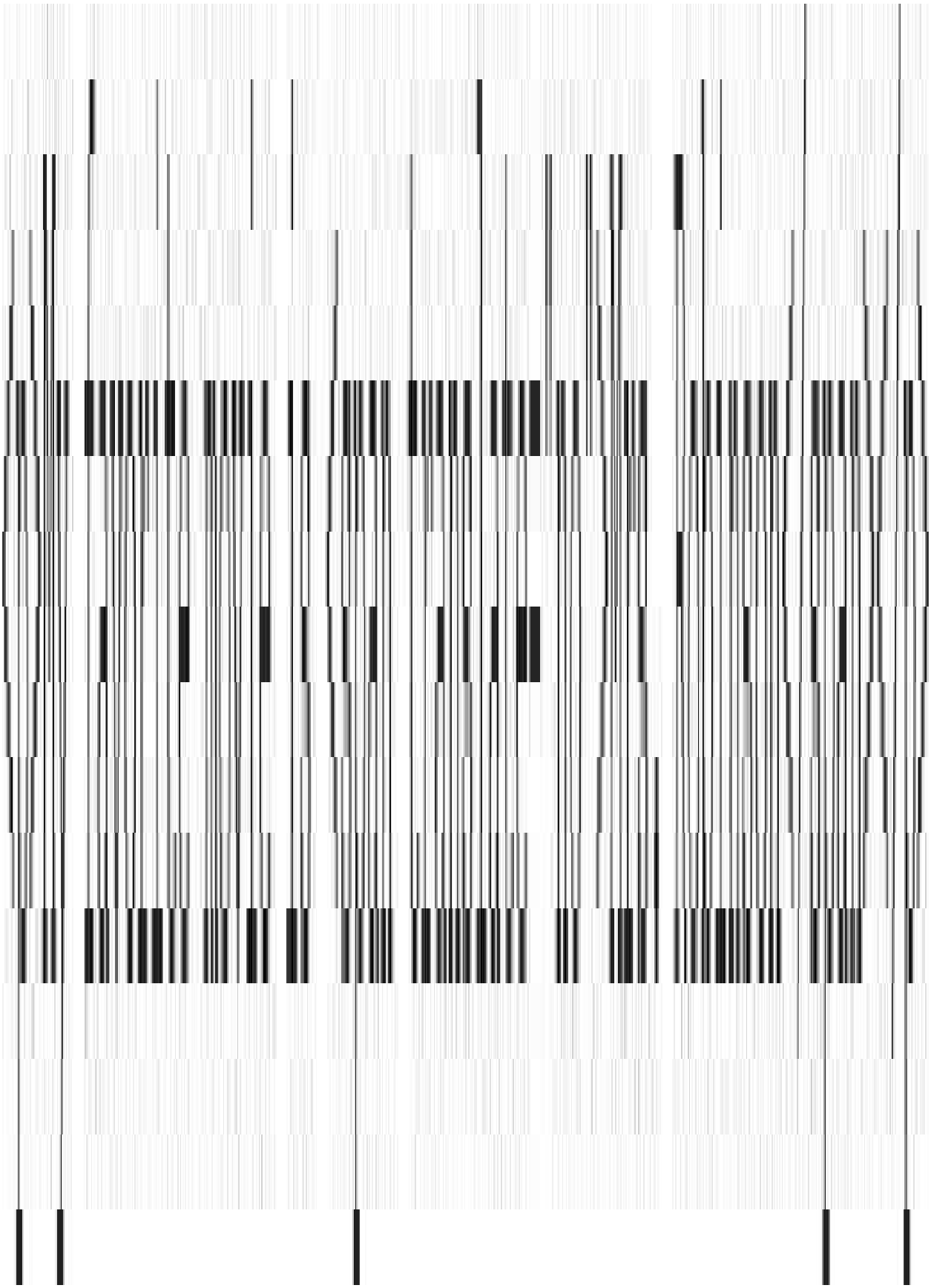
En el ejemplo del Código 1-31, la URL del hipervínculo apunta al archivo myfile.pdf. En circunstancias normales, un navegador moderno mostraría el contenido del archivo en la pantalla, pero en este caso el atributo download fuerza al navegador a descargar el archivo. También incluimos un atributo ping que apunta a un archivo llamado control.php. Como resultado, cada vez que un usuario hace clic en el enlace, el archivo PDF se descarga y el script PHP es ejecuado, lo que permite al desarrollador hacer un seguimiento de esta actividad (es posible almacenar información del usuario en una base de datos, por ejemplo).

Hágalo usted mismo

Copie el Código 1-31 en un archivo de texto, guárdelo con un nombre y la extensión .html, y abra el archivo en su navegador. Al hacer clic en el enlace, el navegador le pedirá autorización para descargar el archivo. Luego elimine el atributo download y compruebe la diferencia.

1.7.4 translate

La mayoría de los navegadores actuales traducen automáticamente cualquier documento en el que detectan lengua extrajera, pero en algunos casos hay textos que deben permanecer intactos, como pueden ser nombres, títulos originales, etc. El atributo translate se añade para controlar el proceso de traducción desde HTML.



Codigo 1-32

Uso el atributo translate.

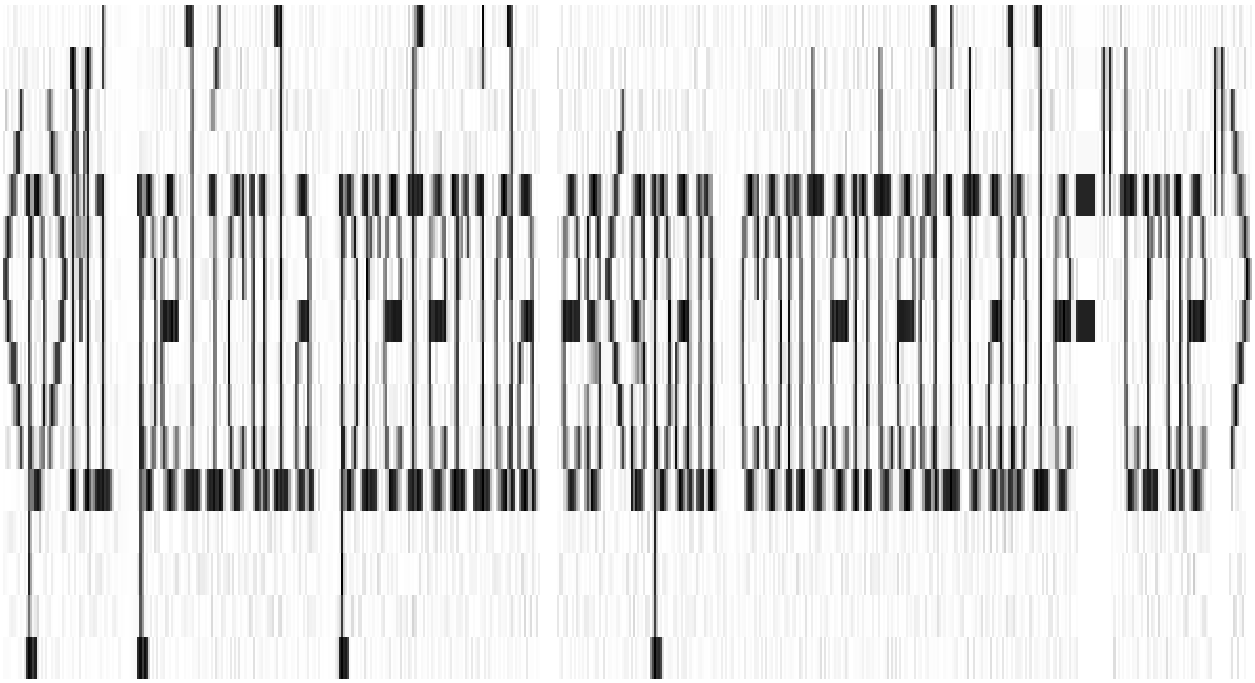
El atributo translate puede tener dos valores: yes (sí) o no (no). Por defecto, el valor es yes, salvo que un elemento de mayor jerarquía haya sido cambiado a no. En el ejemplo del Código 1-32, el nombre de la película está rodeado por etiquetas para especificar que esta parte del texto debe ser preservada tal como se presenta.

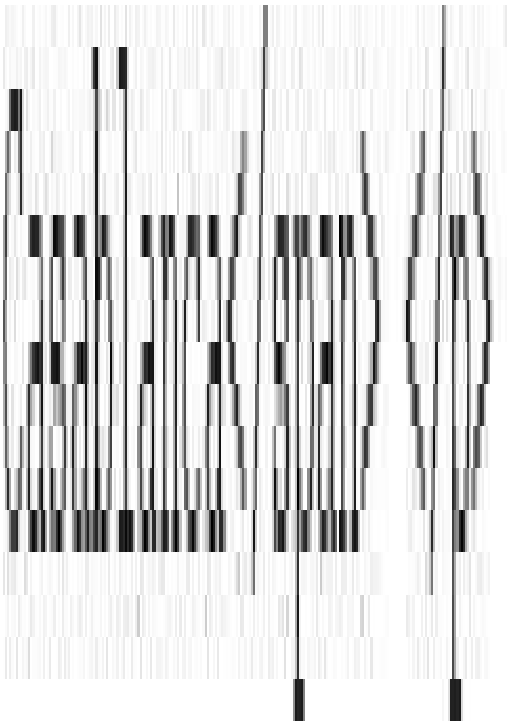
Conceptos básicos

** es un elemento genérico usado para los propósitos organizacionales o de estilo. Debido a sus características, se utiliza a menudo para aplicar estilo al texto, en sustitución del tradicional elemento <font>, actualmente en desuso. Este elemento comparte características similares con el elemento , con la diferencia de que este último es un elemento de bloque mientras es un elemento de línea.**

1.7.5 contenteditable

La posibilidad de editar el contenido de una página web ha existido durante años, pero nunca antes había sido una característica oficial. Ahora, con la introducción del atributo contenteditable, podemos convertir cualquier elemento HTML en uno editable.





Código 1-33

Uso de contenteditable para editar texto.

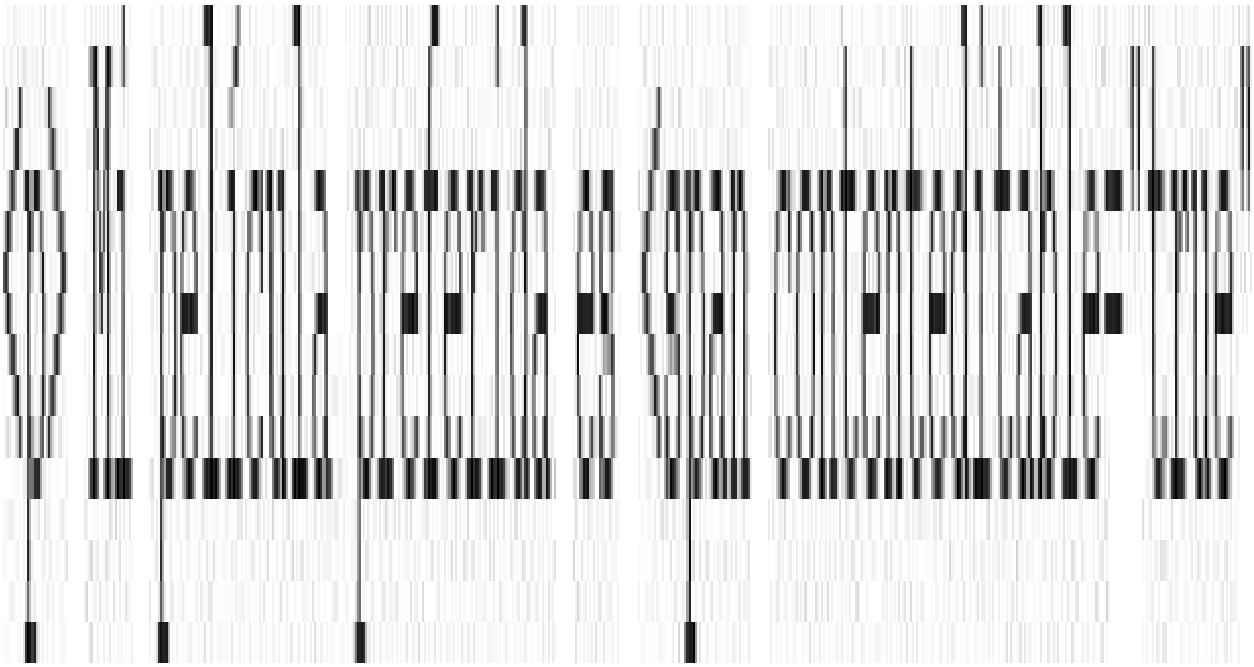
Ese atributo admite dos valores posibles: true (verdadero) o false (falso). Por defecto ningún elemento se puede editar. Al establecer el atributo como true para un elemento, se permite a los usuarios editar su contenido.

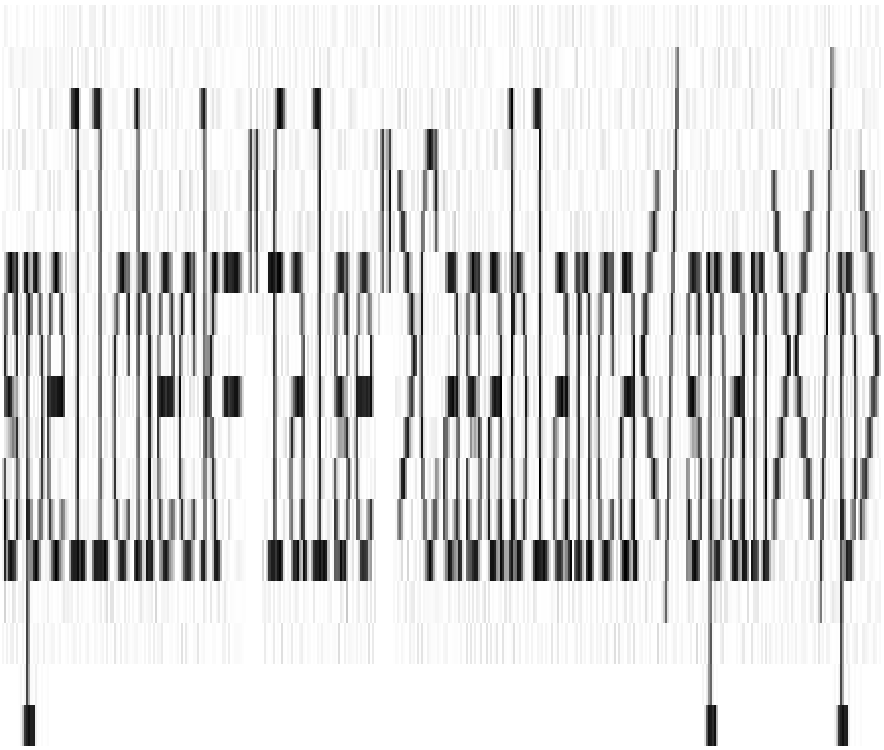
Hágalo usted mismo

Copie el Código 1-33 en un archivo vacío, guárdelo con un nombre y la extensión .html, y abra el archivo en su navegador. Finalmente, haga clic en el nombre de la película para cambiarlo.

1.7.6 spellcheck

Otra característica que se activa automáticamente en los navegadores es la revisión ortográfica. Si bien ésta es una herramienta útil con la que los usuarios esperan contar todo el tiempo, puede ser inapropiada en algunas circunstancias. Para activar o desactivar esta característica se puede utilizar el atributo Spellcheck.





Código 1-34

Desactivación de la función Spellcheck.

Este atributo, igual que sucedía con los anteriores, también tiene dos valores posibles: true o false. La función siempre está activada a menos que se declare el valor de este atributo como false (falso). En el ejemplo en el Código 1-34, el atributo de corrección ortográfica se aplica al mismo elemento que hemos utilizado antes. Ahora, el usuario podría cambiar el nombre de la película sin que el navegador realice una comprobación de errores ortográficos o gramaticales.

2. Estilos CSS y modelos de caja

2.1 CSS y HTML

Como se ha expuesto anteriormente, la nueva especificación de HTML no solo se refiere a los elementos y al lenguaje HTML en sí mismo. La Web exige diseño y funcionalidad, no solo organización estructural y definiciones de secciones. En este nuevo paradigma, HTML se fusiona con CSS y Javascript como un instrumento integrado. Hasta ahora hemos considerado la función de cada tecnología y hemos estudiado los nuevos elementos HTML que son responsables de la estructura del documento. Ahora vamos a examinar la relevancia de CSS en esta unión estratégica y su influencia en la presentación de documentos HTML.

Importante

En este capítulo presentamos una breve introducción a los estilos CSS. Solo se mencionan las técnicas y las propiedades necesarias para la comprensión de los ejemplos en este libro. Si no tiene ninguna experiencia con CSS, le recomendamos seguir algunos tutoriales de la Web. Para encontrar estos recursos, visite nuestro sitio web y siga los enlaces presentados para este capítulo.

Oficialmente, CSS no tiene nada que ver con HTML5. CSS no es parte del código y nunca lo fue: de hecho era un complemento desarrollado para superar las limitaciones y reducir la complejidad de HTML. Inicialmente, los atributos dentro de las etiquetas HTML proporcionaban algunos estilos esenciales a los elementos, pero a medida que el lenguaje evolucionó, el código se hizo más complicado de escribir y mantener y HTML por sí mismo ya no era capaz de

satisfacer las demandas de los diseñadores web. Como resultado, pronto se adoptó CSS como la forma de separar estructura y presentación. Desde entonces, CSS ha prosperado y se ha desarrollado de forma paralela, enfocado a los diseñadores y sus necesidades y no necesariamente como una parte de la evolución de HTML.

La nueva versión de CSS, CSS3, sigue el mismo camino pero esta vez con un compromiso mucho mayor. En el desarrollo de la especificación de HTML5 se ha considerado implícitamente que el lenguaje CSS está cargo del diseño. Debido a esto, la integración entre HTML y CSS3 ahora es vital para el desarrollo web, y es por eso que cada vez que mencionamos HTML5 también hacemos referencia al lenguaje CSS3, aun cuando oficialmente se trata de dos tecnologías independientes.

Actualmente, características de CSS3 son aplicadas e incorporadas a navegadores compatibles con HTML5 junto al resto de la especificación. En este capítulo estudiaremos los conceptos básicos de CSS y las nuevas técnicas de CSS3 disponibles. También vamos a aprender acerca de nuevos selectores y seudo-clases que facilitan la selección e identificación de los elementos HTML.

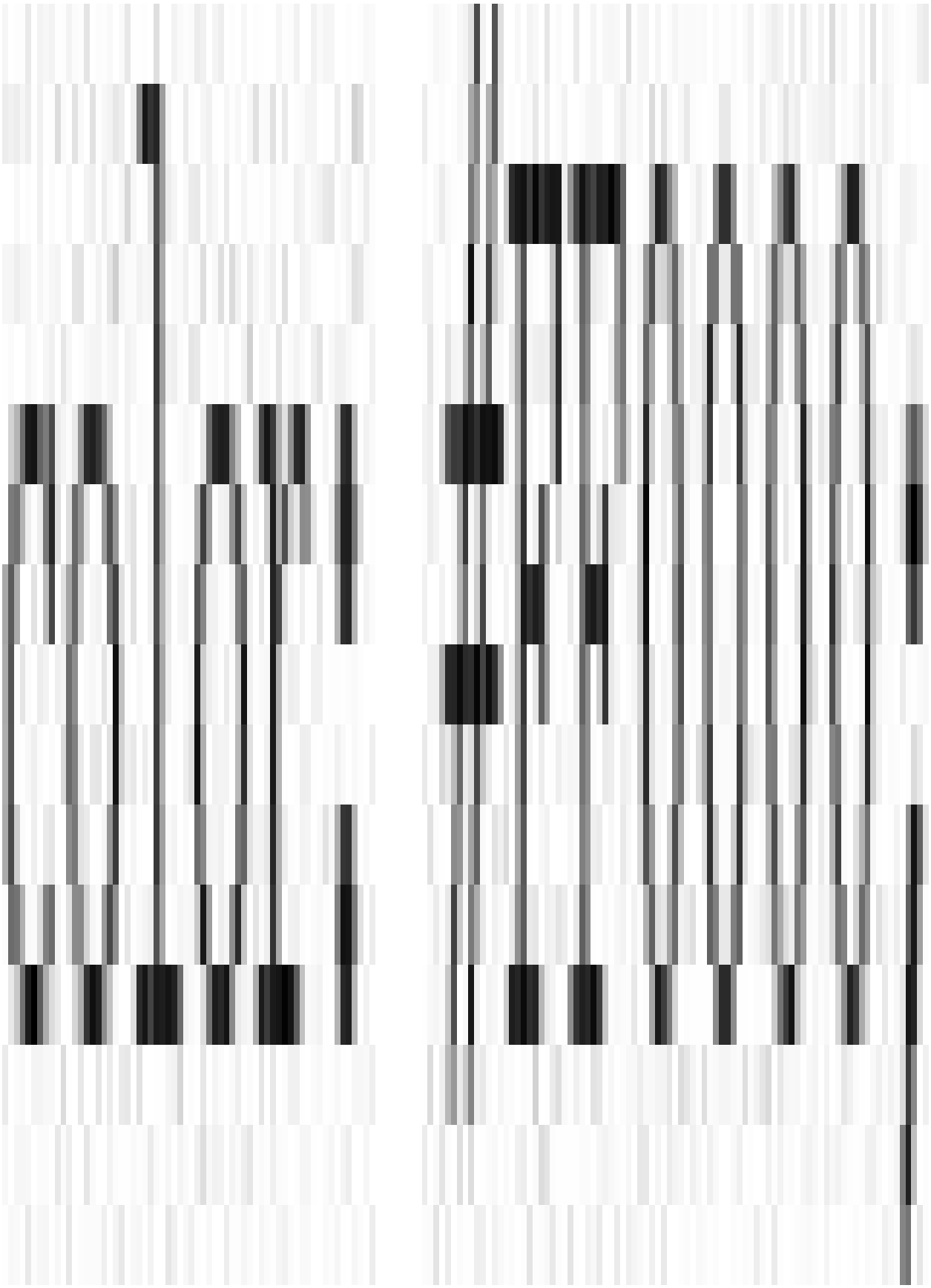
2.2 Breve introducción a CSS

CSS es un lenguaje que trabaja con HTML para aplicar estilos visuales a los elementos del documento, como tamaño, color, fondos, bordes, etc. Aunque cada navegador otorga estilos predeterminados a los elementos HTML, estos pueden no coincidir con la visión del diseñador. De hecho, por lo general están muy lejos de lo que se quiere para el sitio web. Diseñadores y desarrolladores a menudo aplican sus propios estilos para lograr la apariencia y la organización que desean obtener en pantalla.

En esta parte del capítulo vamos a revisar el estilo CSS y a explicar las técnicas básicas para definir la estructura de un documento.

2.2.1 Reglas CSS

CSS define básicamente cómo se van a mostrar en la pantalla los elementos HTML. Para aplicar los estilos, CSS utiliza propiedades y valores. Esta construcción se llama declaración y la sintaxis incluye dos puntos después del nombre de la propiedad y un punto y coma para cerrar la línea.



Código 2-1

Declaración de propiedades CSS.

En el ejemplo del Código 2-1, el nombre de la propiedad es color y el valor asignado a esta propiedad es #FF0000. Si esta propiedad se aplica posteriormente a un elemento HTML, el elemento se mostrará en la pantalla en color rojo.

Importante

Algunos de los conceptos explicados en los párrafos siguientes son para introducir a principiantes en el lenguaje CSS y los conceptos básicos de diseño. Si está familiarizado con esta información, por favor siéntase libre de saltar los apartados que ya conoce.

Conceptos básicos

En CSS, los colores se definen por la combinación de tres colores básicos: rojo, verde y azul. Para representar los colores, podemos utilizar números hexadecimales (de 00 a FF) o números decimales (de 0 a 255). Si decidimos utilizar números hexadecimales, tenemos que expresar el color con una almohadilla (#) al comienzo, como en el ejemplo del Código 2-1. Para los

números decimales hay una función disponible llamada rgb y los colores se declaran usando la sintaxis rgb(255, 0, 0). Vamos a estudiar la función rgb() y otras funciones similares más adelante en este libro.

Las propiedades pueden ser agrupadas usando llaves ({}). Este grupo de una o más propiedades se denomina regla y es identificado por un nombre o un selector, que representa el elemento o grupo de elementos que se verán afectados por la regla. Podemos crear tantas reglas como queramos.

```
p {
    color: #FF0000;
    font-size: 24px;
}
```

Código 2-2

Declaración de reglas CSS.

En el Código 2-2 usamos dos propiedades con sus correspondientes valores entre llaves (color y font-size). Esta regla se identifica con el nombre p. En este caso, el nombre de la regla es una referencia a los elementos <p> del documento. Si aplicamos esta regla a nuestro documento, el contenido de cada elemento <p> será de color rojo y con un tamaño de 24 px. Es posible declarar tantos selectores como se desee en una regla CSS, solo se necesita escribir sus nombres separados por una coma:

```
p, span {
    color: #FF0000;
    font-size: 24px;
}
```

Código 2-3

Declaración de reglas CSS.

En el ejemplo del Código 2-3, la norma afecta a todos los elementos <p> y encontrados en el documento

De modo similar, también podemos mencionar solo los elementos que estén dentro de un elemento en particular enumerando los selectores separados por un espacio. Por ejemplo, los elementos contenidos dentro de elementos <p>:

```
p span {
    color: #FF0000;
    font-size: 24px;
}
```

Código 2-4

Declaración de reglas CSS.

Existen diferentes métodos para hacer referencia a elementos HTML desde CSS. La que se aplica en los ejemplos anteriores simplemente utiliza el nombre del tipo de elementos que deseamos ser afectados. Ésta es una referencia general, pero hay otras más específicas que estudiaremos en breve.

2.2.2 Propiedades

Como podrá ver, las propiedades son el núcleo del lenguaje CSS. Hay docenas de propiedades y en cada versión del lenguaje se añaden otras propiedades nuevas. Vamos a aplicar algunas de esas propiedades en situaciones prácticas a lo largo de todo el libro. Mientras tanto, aquí hay una lista de las propiedades más comunes (incluidas desde la primera versión de CSS) que encontrará en nuestros ejemplos:

font: Permite declarar varios estilos de texto, como ancho, tamaño, familia, etc. Los valores deben estar separados por un espacio y declarados en un orden específico (por ejemplo: font: bold 24px arial, sans-serif;). También podemos declarar todos los estilos de forma independiente utilizando las propiedades asociadas font-style, font-variant, font-weight, font-size, line-height y font-family (por ejemplo, font-size: 24px).

color: Esta propiedad declara el color de un elemento. El valor se puede expresar en números hexadecimales (por ejemplo, color: #FF0000;) o en números decimales (por ejemplo, color: rgb(255, 0, 0);).

background: Permite aplicar varios estilos al fondo de un elemento, como el color, imagen, repetición, etc. Los valores indicados deben estar separados por un espacio (por ejemplo, background: #0000FF url('bricks.jpg') no-repeat;). Cada estilo puede ser declarado de forma independiente utilizando propiedades individuales. Vamos a estudiar esta propiedad y todas las propiedades asociadas a ella con mayor detalle en el Capítulo 3.

width: Declara el ancho de un elemento (por ejemplo, width: 200px).

height: Declara la altura de un elemento (por ejemplo, height: 200px).

margin: Esta propiedad declara el margen externo de un elemento. El margen es el espacio alrededor del elemento. Puede tener cuatro valores: arriba, derecha, abajo y a la izquierda, en ese orden y separados por un espacio (por ejemplo, margin: 10px 30px 10px 30px;). Sin embargo, si solo se declaran uno, dos o tres parámetros, los demás tendrán los mismos valores (por ejemplo, margin: 10px 30px;). Los valores también se pueden declarar de forma independiente utilizando las propiedades asociadas: margin-top (margen superior), margin-right (margen derecho), margin-bottom (margen inferior) y margin-left(margen izquierdo), como en margin-left: 10px;.

padding: Esta propiedad declara el margen interno de un elemento. Se refiere al espacio que rodea el contenido del elemento pero está en el interior de su borde, por ejemplo, el espacio entre el título y el borde de la caja virtual creada por el elemento <h1> que contiene ese título. Los valores se declaran de la misma manera que para la propiedad de margen y, por tanto, también se pueden declarar de forma independiente utilizando las propiedades asociadas padding-top, padding-right, padding-bottom y padding-left (por ejemplo, padding-top: 10px;).

border: Declara el ancho, estilo y color del borde de un elemento (como por ejemplo, border: 1px solid #990000;). Los valores posibles para el estilo son: none, hidden, dotted, dashed, solid, double, groove, ridge, inset, outset o inherit. Esta propiedad también tiene propiedades asociadas para cada parte de un elemento: border-top, border-bottom, border-left and border-right. También podemos asignar valores para el ancho, estilo y color para cada lado de forma independiente añadiendo el nombre al final de la propiedad, pero siempre es necesario declarar la propiedad border antes de hacer cualquier cambio a la misma (por ejemplo, border-style: solid;

border-top-width: 10px;).

text-align: esta propiedad alinea el elemento dentro del elemento que lo contiene. Los valores posibles son left, right, center, justify o inherit.

Importante

Estas no son las únicas propiedades disponibles de CSS. En este libro vamos a aplicar solo algunas de las propiedades tradicionales y, además, estudiaremos las nuevas propiedades incorporadas en CSS3. Para aprender más sobre este lenguaje, por favor visite nuestro sitio web y siga los enlaces de este capítulo.

2.2.3 Estilos en línea

Al aplicar estilos a los elementos HTML cambia la forma en la que estos se presentan en pantalla. Como dijimos antes, los navegadores proporcionan un conjunto de estilos por defecto que en la mayoría de los casos no se ajustan a las necesidades de los diseñadores. Para cambiar esto, podemos sobrescribirlos con nuestros propios estilos utilizando diferentes técnicas. Una de estas técnicas es asignar estilos dentro del elemento como un atributo. El Código 2-5, presentado a continuación contiene un sencillo documento HTML que muestra el elemento <p> modificado por el atributo style con el valor font-size: 20px. Esta propiedad cambia el tamaño predeterminado del texto dentro del elemento <p> a 20 px.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Este texto es el título de mi documento</title>
</head>
<body>
  <p style="font-size: 20px">Mi texto</p>
</body>
</html>
```

Código 2-5

Estilos CSS dentro de las etiquetas HTML.

Hágalo usted mismo

Copie en un archivo de texto el código y ábralo en su navegador para comprobar su funcionamiento. Tenga en cuenta que el archivo debe tener la extensión .html para abrirlo correctamente (por ejemplo, miarchivo.html).

Usar la técnica descrita anteriormente es una buena manera de probar estilos y ver sus efectos, pero no se recomienda para un proyecto más grande. La razón es simple: con esta técnica es necesario escribir y repetir cada estilo para cada elemento, aumentando así considerablemente el tamaño del documento. De este modo se acaba haciendo imposible actualizar y mantener el sitio. Imagínese que decide que en vez de 20 px, el tamaño del texto en cada elemento <p> debe ser de 24 px. Tendría que cambiar cada estilo en cada etiqueta <p> en todo el documento y en cada uno de los documentos de su sitio web.

2.2.4 Estilos incrustados

Mejor alternativa a la técnica anterior es la inserción de estilos en el encabezado del documento y el uso de referencias para aplicarlos a los elementos HTML correctos.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Este texto es el título de mi documento</title>
  <style>
    p { font-size: 20px }
  </style>
</head>
<body>
  <p>Mi texto</p>
</body>
</html>
```

Código 2-6

Código de estilos en el encabezado del documento.

El elemento <style> permite a los desarrolladores insertar los estilos CSS en una página. En las versiones anteriores de HTML era necesario especificar qué tipo de estilos se insertarían pero en HTML5 el estilo por defecto es CSS, así que no es necesario añadir ningún atributo a la etiqueta de apertura <style>.

Las líneas en negrita en el Código 2-6 cumplen la misma función que la línea en negrita en el Código 2-5, pero en este ejemplo no es necesario escribir el estilo dentro de cada elemento <p> del documento porque ya se ha indicado su aplicación a todos ellos. Con este método, se ha reducido código y se ha asignado el estilo deseado utilizando referencias a los elementos adecuados.

2.2.5 Archivos externos

La declaración de los estilos en el encabezado del documento ahorra espacio y hace que el código sea más consistente y fácil de mantener, pero se requiere una copia de los estilos en todos los documentos de nuestro sitio web. La solución a esta duplicación es mover todos los estilos en un archivo externo y utilizar el elemento <link> para insertar este archivo en cualquier documento que requiera la aplicación del estilo. Este método también permite cambiar todo un conjunto de estilos mediante la simple inserción de un archivo diferente, así como modificar o adaptar los documentos a cada circunstancia o dispositivo, tal como veremos al final del libro.

En el Capítulo 1 se estudió la etiqueta <link> y cómo insertar archivos CSS en nuestros documentos. Usando de la línea <link rel=“stylesheet” href=“misestilos.css”>, indicamos al navegador que cargase el archivo misestilos.css, que contiene todos los estilos necesarios para procesar la página. Esta práctica es ampliamente utilizada entre los diseñadores que trabajan con HTML5. La etiqueta <link> referida al archivo CSS se puede insertar en cualquier documento en el que sean pertinentes los estilos que contiene.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Este texto es el título de mi documento</title>
  <link rel="stylesheet" href="misestilos.css">
</head>
<body>
  <p>Mi texto</p>
</body>
</html>
```

Código 2-7

Aplicación de estilos CSS desde un archivo externo.

Hágalo usted mismo

A partir de ahora añada los estilos CSS en un archivo llamado misestilos.css. Cree ese archivo en la misma carpeta que el archivo HTML y copie los estilos CSS dentro de este archivo para ver la forma en la que trabajan.

2.2.6 Referencias

Poner todos los estilos en un solo archivo externo y vincular ese archivo desde todos los documentos es muy conveniente. Sin embargo, necesitamos mecanismos para establecer una relación específica entre estos estilos y los elementos del documento que se verán afectados por ellos.

En los ejemplos anteriores se aplicó una de las técnicas que se utilizan para asignar estilos a un elemento. Por ejemplo, en el Código 2-6, el estilo usado para cambiar el tamaño de la fuente hacía referencia a cada elemento <p> usando la palabra clave p. Como resultado, el estilo CSS insertado entre las etiquetas <style> se asignó a cada elemento <p> del documento. Existen tres métodos básicos para seleccionar qué elemento HTML se verá afectado por una regla de CSS:

Mediante la palabra clave del elemento.

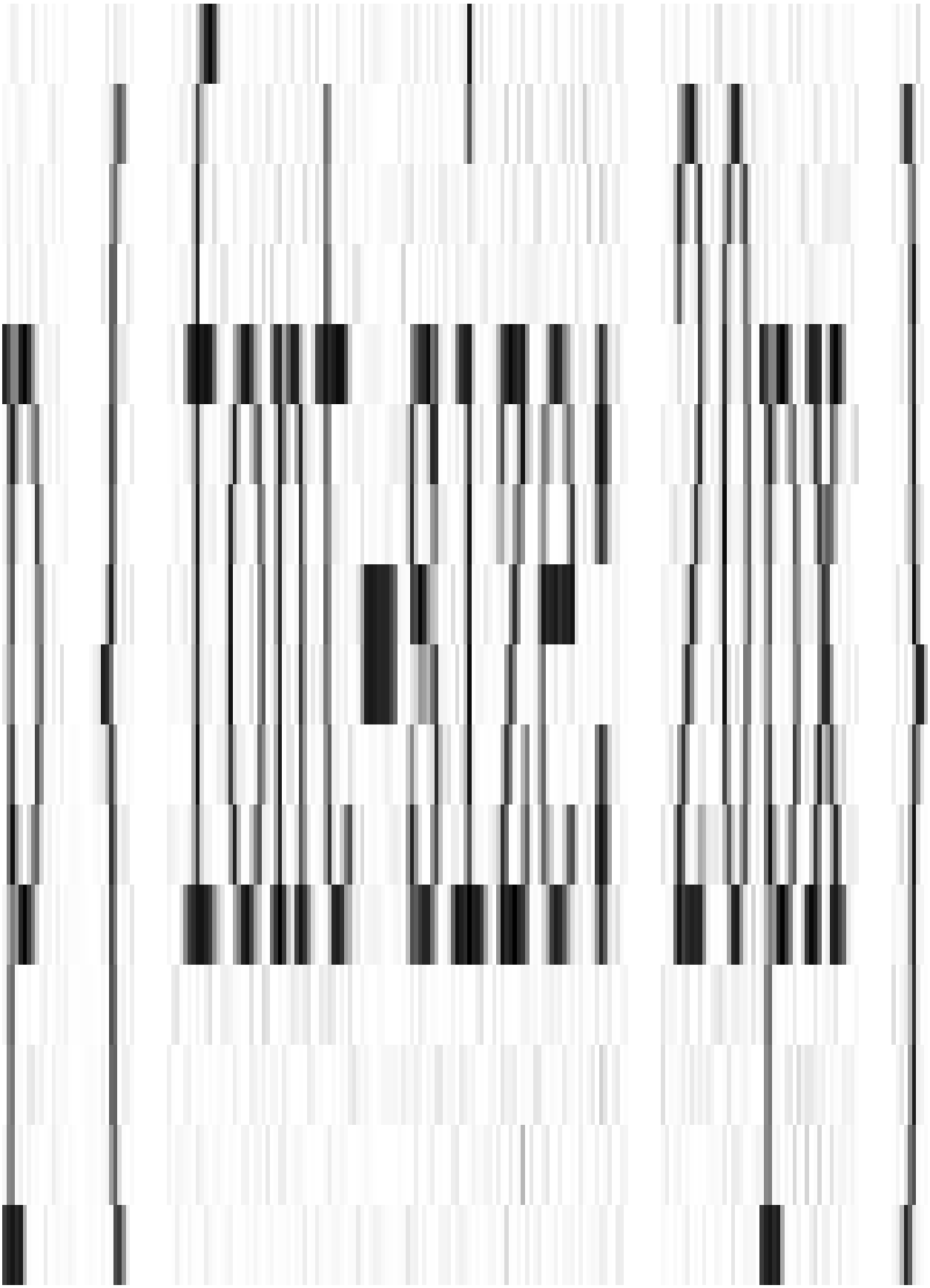
Mediante el atributo id.

Mediante el atributo class.

Sin embargo, ya veremos más adelante que CSS3 es flexible en este sentido e incorpora nuevas formas de hacerlo que resultan mucho más específicas.

Referencias por palabra clave

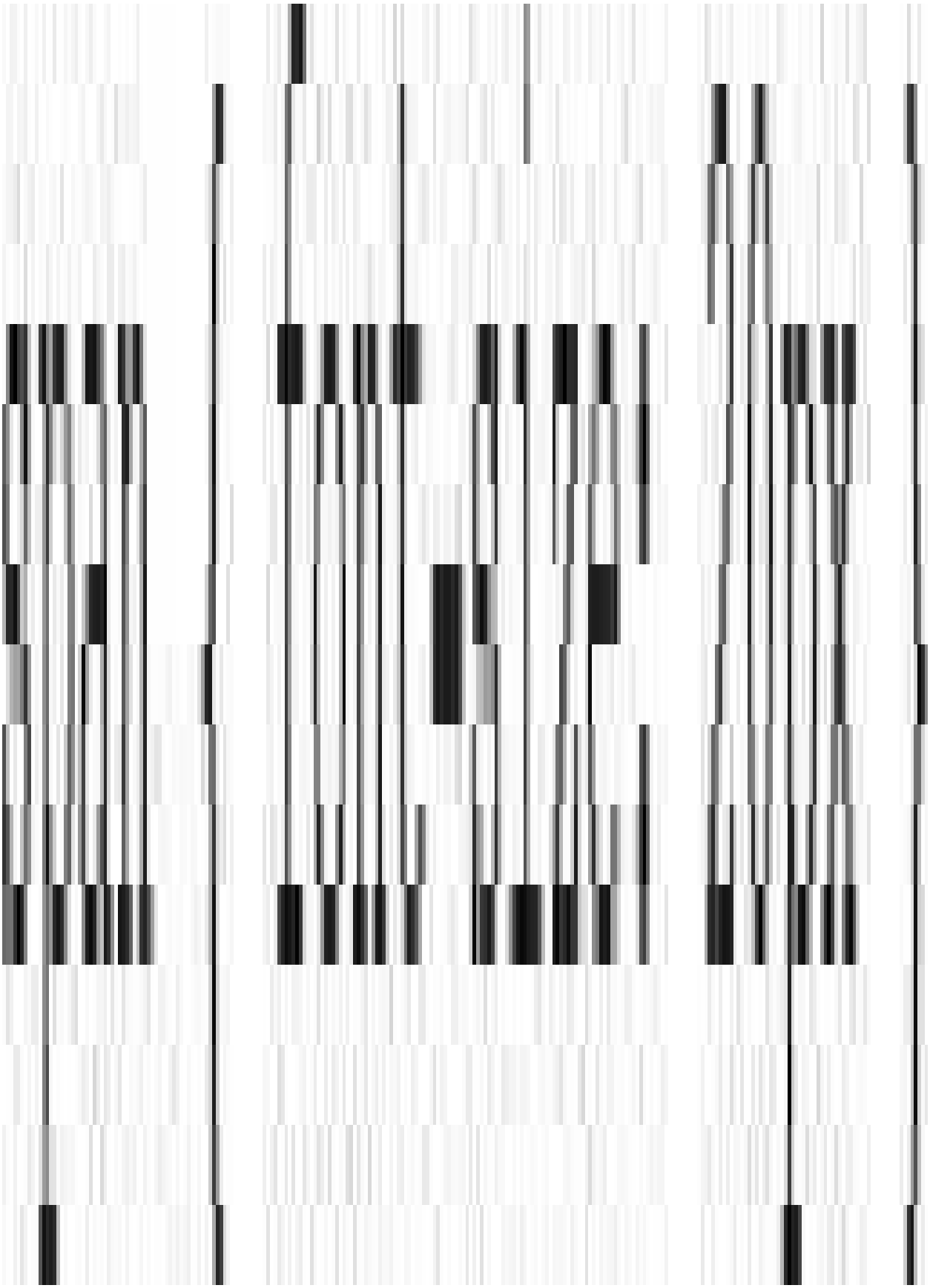
Al declarar de la regla CSS con la palabra clave del elemento ésta afectará a todos los elementos del mismo tipo en el documento. Por ejemplo, la siguiente regla cambiará los estilos de los elementos <p>:



Código 2-8

Referencias por palabra clave.

Al colocar la clave p frente a la regla, le indicamos al navegador que esta regla se debe aplicar a cada elemento <p> del documento HTML. En consecuencia, todos los textos encerrados entre etiquetas por <p> tendrán un tamaño de 20 px. Por supuesto, el mismo principio funcionará para cualquier otro elemento HTML en el documento. Si se especifica la palabra clave span en lugar de p, por ejemplo, cada texto entre etiquetas tendrá un tamaño de 20 px.



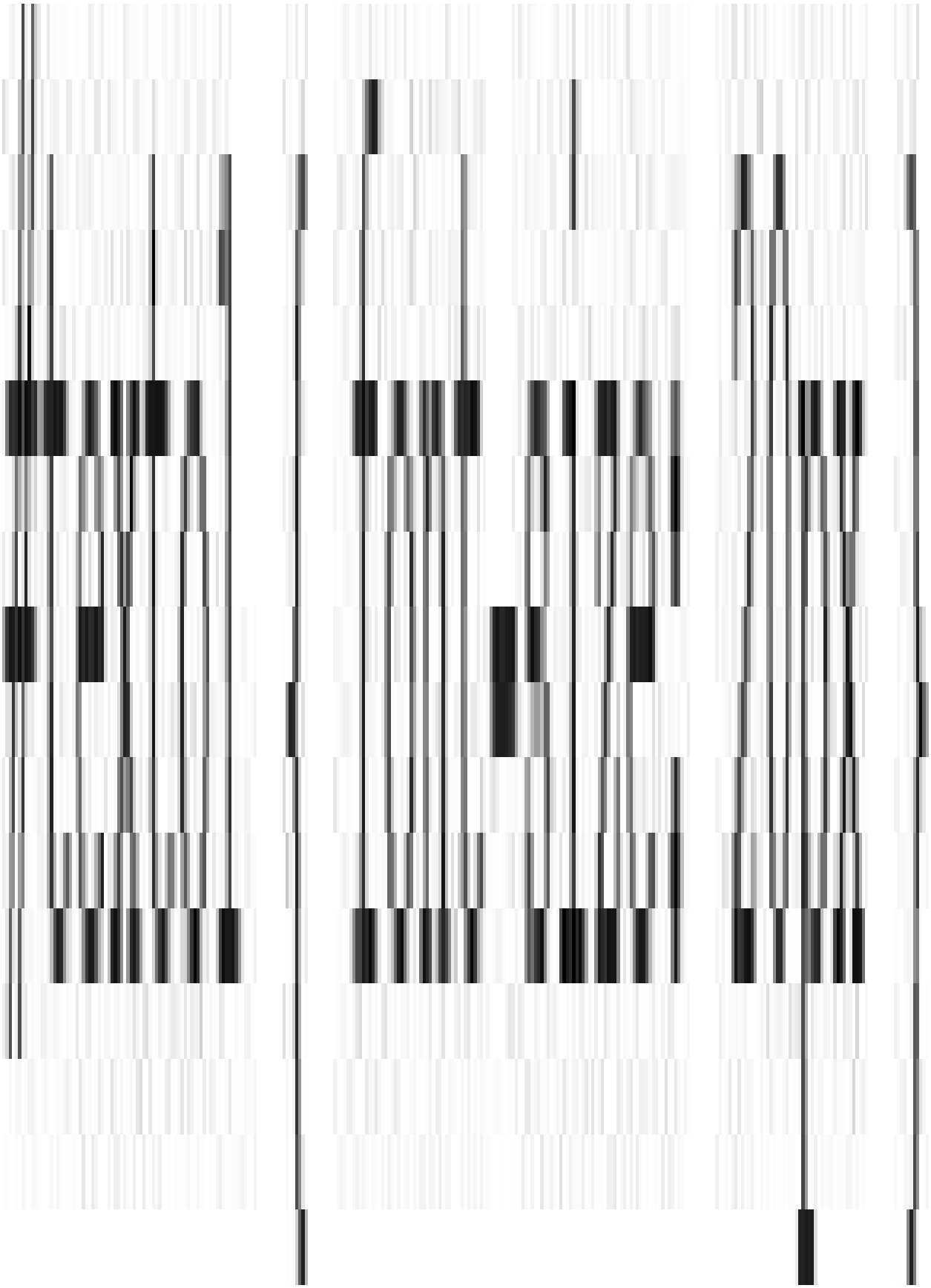
Código 2-9

Referencia por otra palabra clave.

Pero, ¿qué pasa si solo se necesita hacer referencia a una etiqueta específica? ¿Es necesario utilizar el atributo de estilo dentro de esa etiqueta? La respuesta es no. Como hemos comentado antes, el método de estilos en línea (aplicado mediante el atributo style ubicado dentro de las etiquetas HTML) es una técnica en desuso, y su uso debe reservarse solo para propósitos de prueba. Para seleccionar un elemento específico del código HTML de las reglas en nuestro archivo CSS, podemos utilizar alguno de estos dos atributos: id y clase.

Referencias con el atributo id

El atributo id es más bien un nombre, una identificación del elemento. Por este motivo el valor de este atributo no puede ser duplicado, pues perdería su sentido: este nombre debe ser único en todo el documento. Para hacer referencia a un elemento en particular del documento HTML mediante el atributo id desde nuestro archivo CSS, la regla debe ser declarada con el signo almohadilla (#) delante del valor de identificación.



Código 2-10

Crear referencia mediante el valor del atributo id.

La regla en el Código 2-10 se aplica al elemento HTML identificado por el atributo id=“texto1”. Ahora podemos modificar nuestro documento HTML de ejemplo con este código:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Este texto es el título de mi documento</title>
  <link rel="stylesheet" href="misestilos.css">
</head>
<body>
  <p id="texto1">Mi texto</p>
</body>
</html>
```

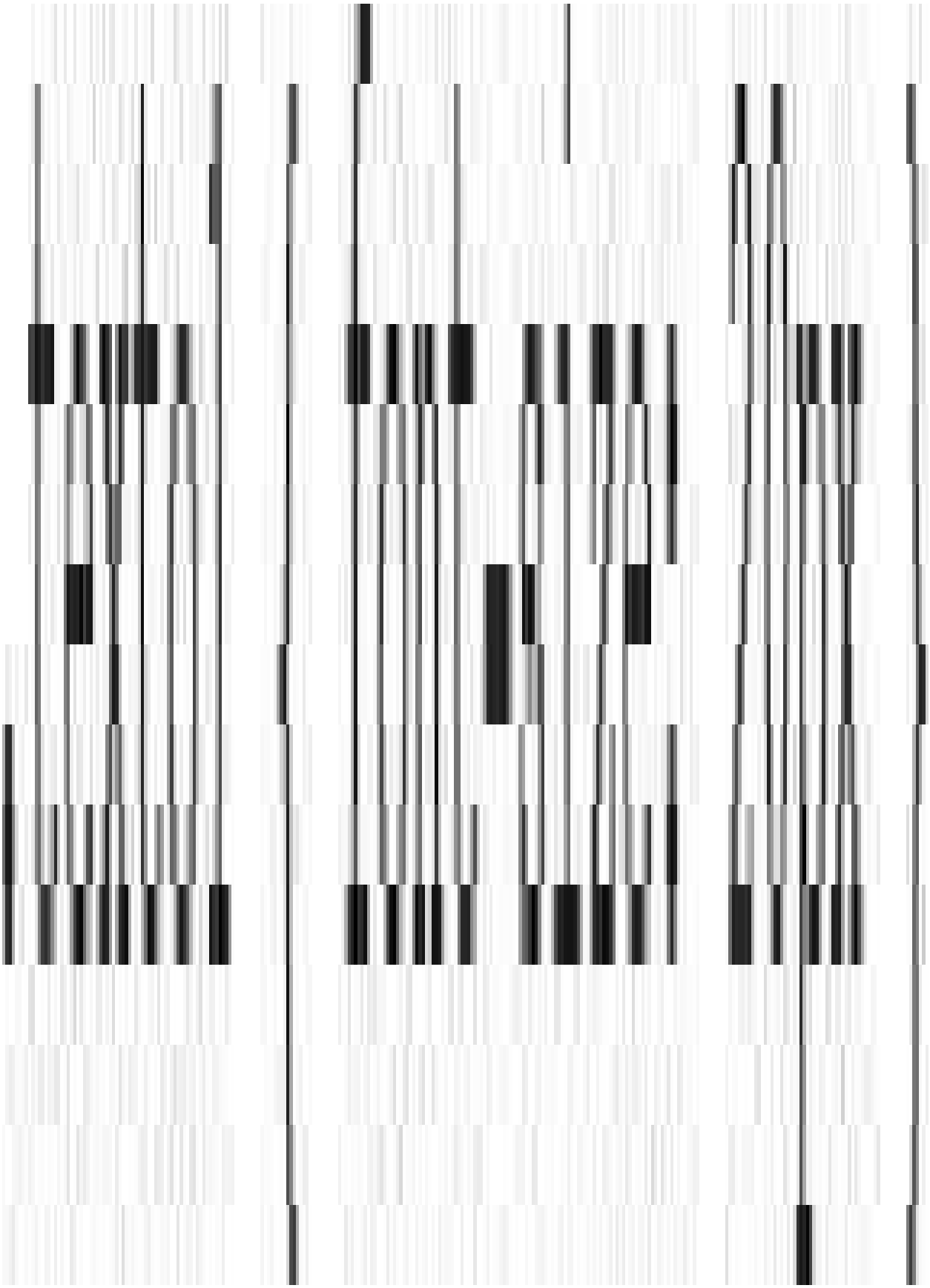
Código 2-11

Identificar el elemento <p> por su atributo id.

El resultado es que cada vez que hacemos una referencia utilizando la identificación texto1 en nuestro archivo CSS, el elemento con ese valor id será modificado, pero el resto de los elementos <p>, o cualquier otro elemento en el documento, no se verá afectado. Es una forma extremadamente específica de hacer referencia a un elemento y se utiliza habitualmente para elementos generales, tales como etiquetas estructurales. De hecho, el atributo id es más adecuado para hacer referencias a los elementos de Javascript, como veremos más adelante.

2.2.6.3 Referencias con el atributo class

En lugar de utilizar el atributo id, la mayor parte de las veces es mejor usar el atributo class para aplicar estilos. Este atributo es más flexible y se puede asignar a cada elemento HTML en el documento que tenga estilos en común.



Código 2-12

Crear referencias por el valor del atributo class.

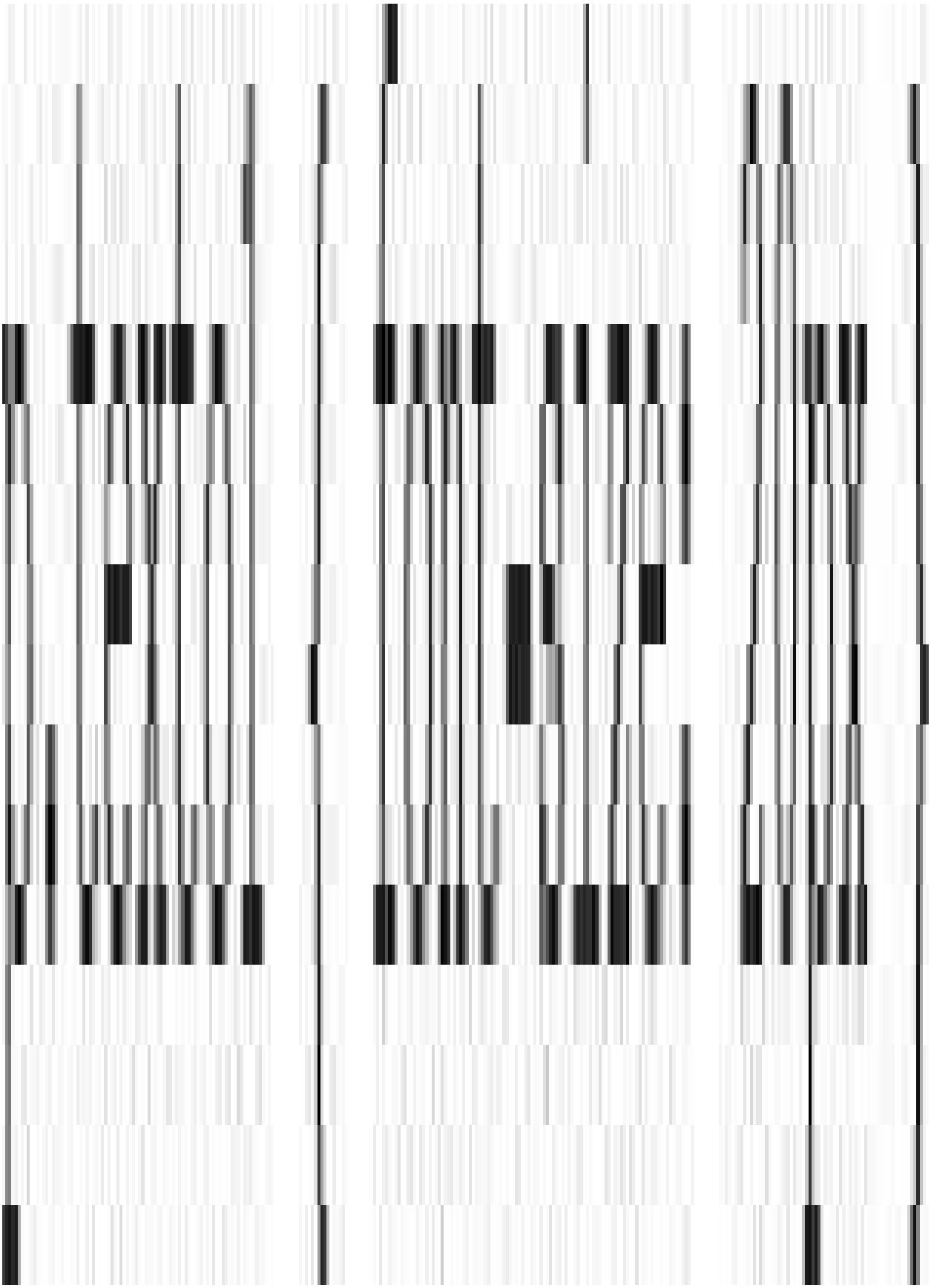
Para trabajar con la clase de atributo, hay que declarar la regla con un punto antes de su nombre. La ventaja es que de este modo la inserción del atributo clase con el valor texto1 será suficiente para asignar la misma regla a cualquier elemento.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Este texto es el título de mi documento</title>
  <link rel="stylesheet" href="misestilos.css">
</head>
<body>
  <p class="texto1">Mi texto</p>
  <p class="texto1">Mi texto</p>
  <p>Mi texto</p>
</body>
</html>
```

Código 2-13

Asignar estilos a través del atributo class.

Los elementos <p> de las dos primeras líneas del interior del cuerpo en el Código 2-13 tienen el atributo class con el valor texto1. Como se dijo antes, la misma regla se puede aplicar a diferentes elementos en el mismo documento. Por tanto, estos dos primeros elementos comparten los mismos estilos y son modificados por la regla del Código 2-12. El último elemento <p> no se ve afectado y mostrará los estilos por defecto. Las reglas asignadas por el atributo class se llaman clases. A un elemento pueden asignarse varias clases. Todo lo que tenemos que hacer es declarar los nombres de las clases separadas por un espacio (por ejemplo, class= “texto1 texto2”). La clase puede ser declarada exclusiva para ciertos tipos de elementos añadiendo un selector antes del punto.



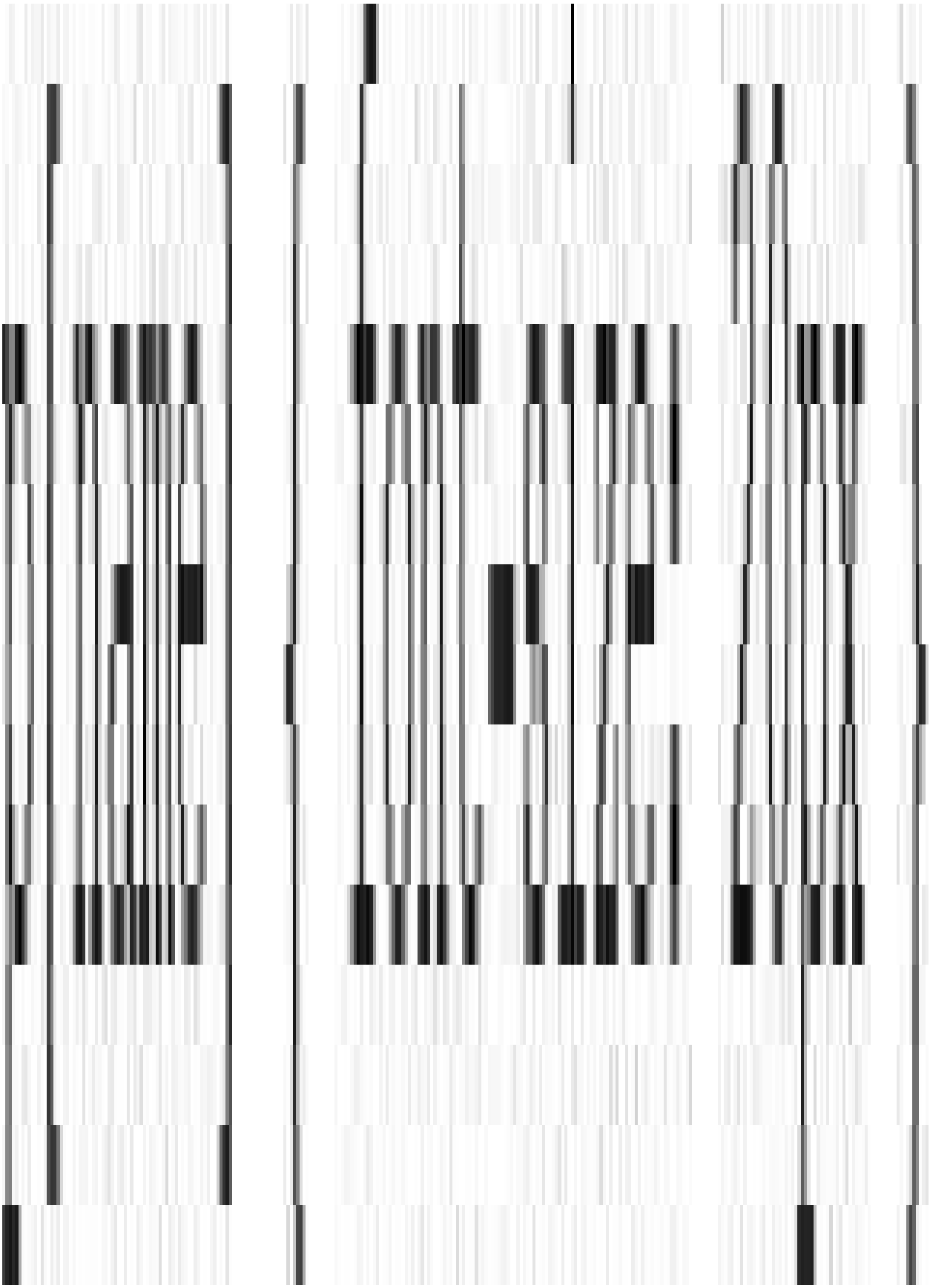
Codigo 2-14

Declarar una clase solo para elementos <p>.

En el Código 2-14 creamos una regla que hace referencia a la clase llamada texto1 pero solo para los elementos <p>. Cualquier otro elemento con el mismo nombre para el atributo class no se verá modificado por esta regla en particular.

2.2.6.4 Referencias con cualquier atributo

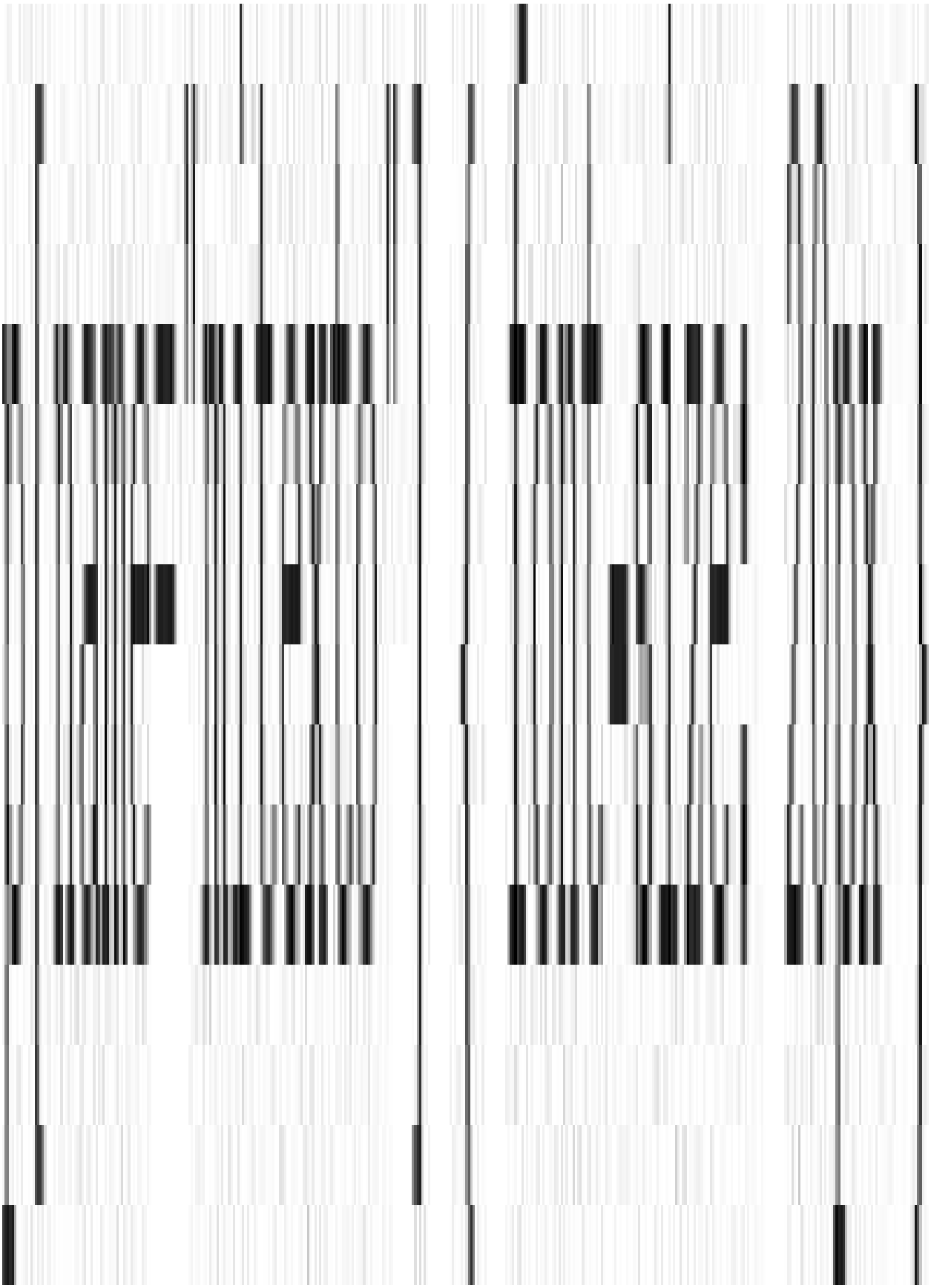
Aunque estos métodos de referencia cubren una variedad de situaciones, a veces no son suficientes para encontrar el elemento exacto al que se desea aplicar un estilo. Las últimas versiones de CSS han incorporado nuevas formas de hacer referencia a los elementos HTML. Uno de ellos es el Selector de atributos. Ahora, podemos hacer referencia a un elemento no solo por sus atributos id y class, sino también por cualquier otro atributo.



Código 2-15

Referenciar elementos <p> que tienen un atributo name.

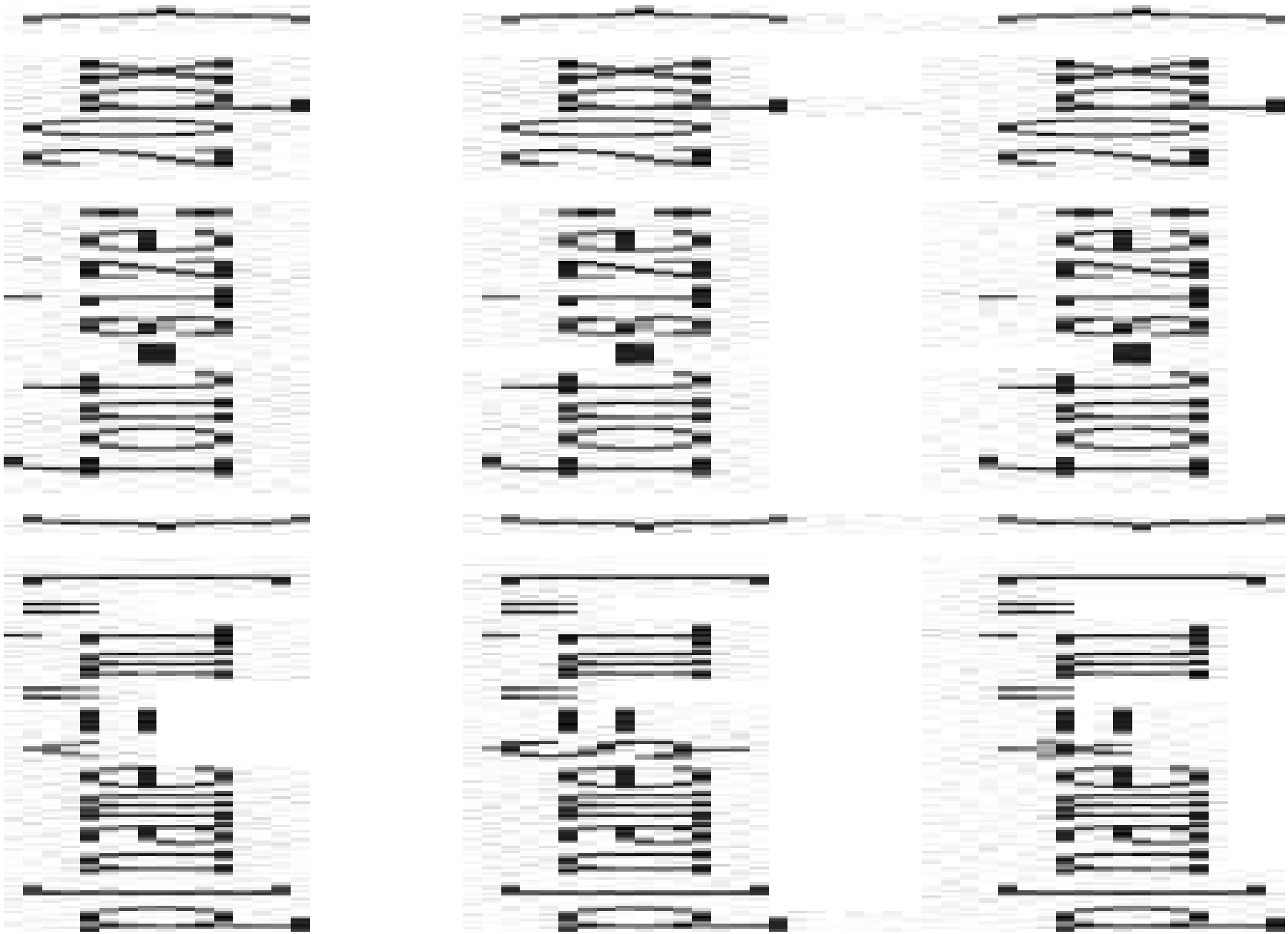
La regla del Código 2-15 solo cambia los elementos <p> que tienen un atributo llamado name. Para reflejar lo que antes hicimos con los atributos id y class, podemos también proporcionar el valor del atributo.



Código 2-16

Referenciar elementos <p> que tienen un atributo name con el valor mitexto.

CSS3 nos permite combinar el símbolo = con otros para hacer una selección más detallada:



Código 2-17

Nuevos selectores en CSS3.

Si ya conoce las expresiones regulares de otras lenguajes como Javascript o PHP, reconocerá los selectores utilizados en el Código 2-17. En CSS3 estos selectores producen resultados similares:

La regla con el selector ^= se asignará a cualquier elemento <p> con un valor de atributo name que comience con “mi” (por ejemplo, mitexto, michoche).

La regla con el selector $= se asignará a cualquier elemento <p> con un valor de atributo de name que termine en “mi” (por ejemplo, textomi, cochemi).

La regla con el selector *= coincidirá con cualquier elemento <p> con un valor del atributo name que contenga la cadena de texto “mi” (en este caso, la cadena de texto también podría estar en el medio, por ejemplo, en textomicoche)

En estos ejemplos, se utiliza el elemento <p>, el atributo name y un texto (“mi”), pero la misma técnica puede ser utilizada con cualquier atributo y el valor que necesite. Para hacer referencia a cualquier elemento HTML, solo tiene que escribir entre corchetes el nombre del atributo y el valor que está buscando.

2.2.6.5 Referencias con pseudo-clases

Las pseudo-clases permiten hacer referencia a elementos HTML por sus características, tales como la posición en el código o las condiciones actuales.

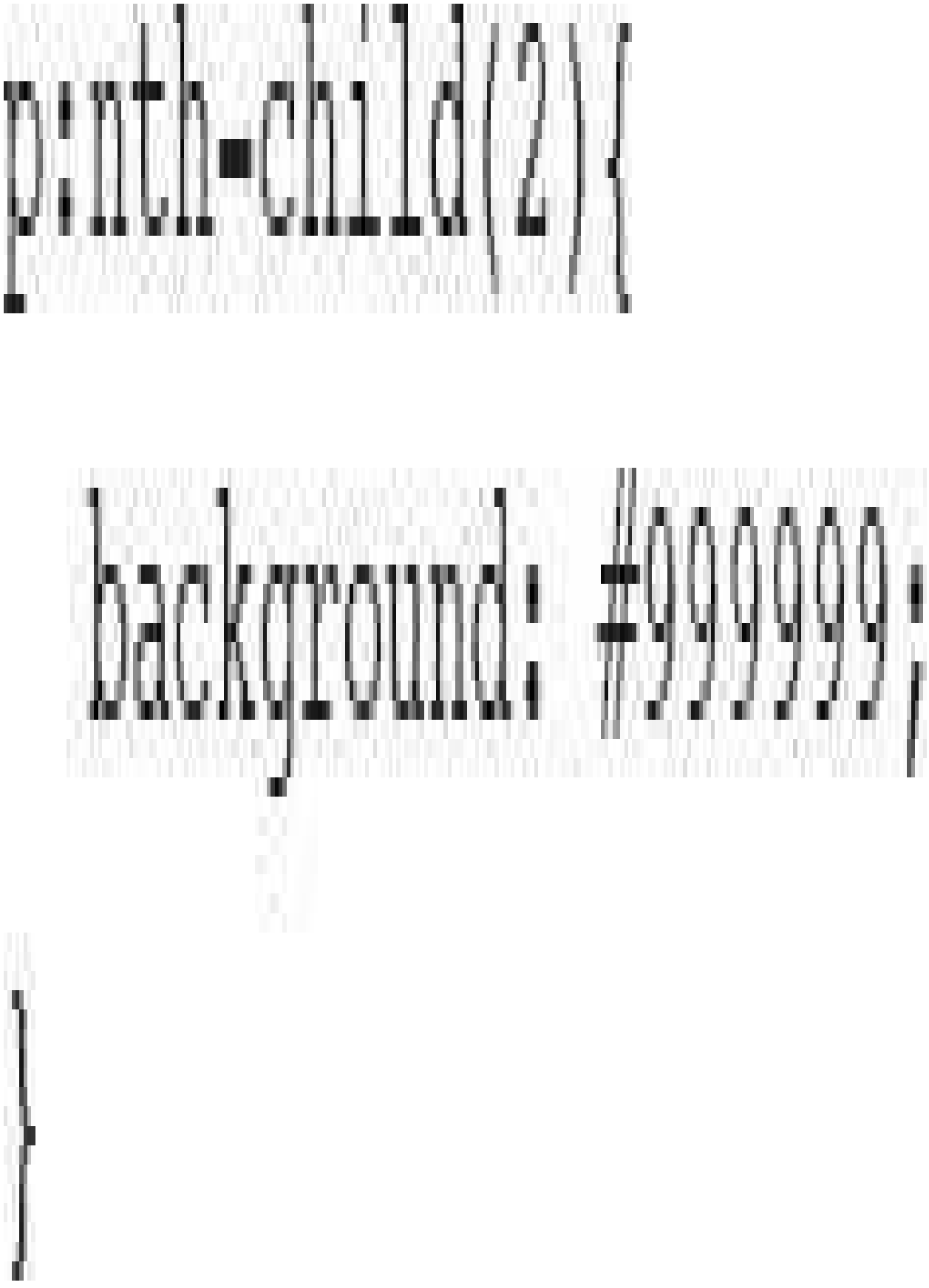
CSS3 incorpora nuevas pseudo-clases que permiten una especificidad aún mayor. Aplicaremos algunas a continuación usando un documento simple y estudiaremos el resto más adelante en situaciones más prácticas.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Éste es el título del documento</title>
  <link rel="stylesheet" href="misestilos.css">
</head>
<body>
  <div id="wrapper">
    <p class="mitexto1">Mi texto1</p>
    <p class="mitexto2">Mi texto2</p>
    <p class="mitexto3">Mi texto3</p>
    <p class="mitexto4">Mi texto4</p>
  </div>
</body>
</html>
```

Código 2-18

Documento HTML para probar pseudo-clases.

Vamos a echar un vistazo al nuevo código HTML (Código 2-18): Tiene cuatro elementos <p> anidados en un mismo elemento *. Utilizando pseudo-clases podemos sacar ventaja de esta organización y hacer referencia a un elemento específico sin importar qué información tengamos sobre sus atributos o valores:


p:nth-child(2){
background: #999999;
}

Código 2-19

Pseudo-clases :nth-child().

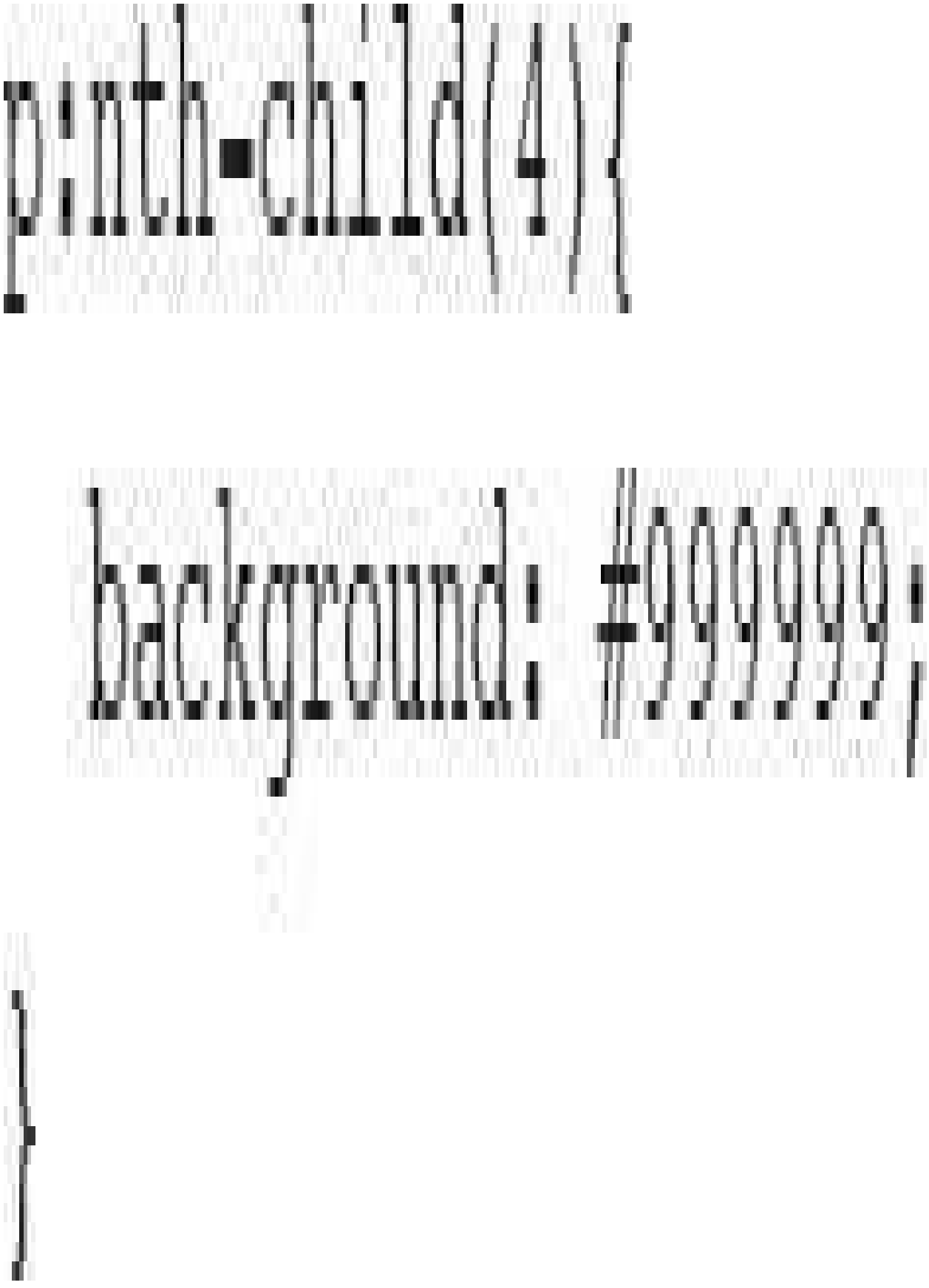
La pseudo-clases se añaden utilizando una coma después de la referencia y antes de su nombre. En la regla del Código 2-19 nos referimos a elementos <p>. Esta regla podría escribirse así: .miclase: nth-child (2) para hacer referencia a cada elemento “hijo” (child) de otro elemento, es decir, anidado en otro elemento, cuyo atributo class tenga un valor igual a miclase. Las pseudo-clases pueden ser añadidas a cualquiera de los tipos de referencias que hemos mencionado hasta ahora.

Hágalo usted mismo

Reemplace el código actual de su archivo HTML por el mostrado en el Código 2-18, incorpore las reglas presentadas en el Código 2-19 al archivo misestilos.css y abra el HTML en el navegador para probar este ejemplo.

La pseudo-clase nth-child () permite ubicar un elemento anidado o hijo específico. Recordará que el documento HTML del Código 2-18 tiene cuatro elementos <p> que son “hermanos”, es decir, que están anidados juntos y, por lo tanto, tienen el mismo “padre” que es el elemento . Lo que esta pseudo-clase indica es algo así como “el elemento hijo en la posición …” por lo que el número entre paréntesis será el número de la posición o índice del elemento. La regla del Código 2-19 hace referencia a cada elemento <p> del documento que ocupe un segundo lugar.

Desde luego, es posible seleccionar cualquier elemento anidado usando el mismo método de referencia pero cambiando el número de índice. Por ejemplo, la siguiente regla tendrá un impacto solo en el último elemento <p>.


p:nth-child(4){
background: #999999;
}

Código 2-20

Pseudo-clases :nth-child().

Como usted probablemente se imaginará, es posible asignar estilos a cada elemento mediante la creación de una regla para cada uno de ellos.

```
  p:nth-child(1){
    background: #999999;
}
p:nth-child(2){
    background: #CCCCCC;
}
p:nth-child(3){
    background: #999999;
}
p:nth-child(4){
    background: #CCCCCC;
}
```

Código 2-21

Creación de una lista con la pseudo-clase :nth-child().

En el Código 2-21 se utiliza la pseudo-clase nth-child() para generar una lista de opciones claramente diferenciadas por el color de fondo.

Hágalo usted mismo

Copie el último código en el archivo CSS y abra el documento HTML para comprobar el efecto en su navegador.

Es posible añadir más opciones a la lista mediante la adición de nuevos elementos <p> en el código HTML y nuevas reglas con la pseudo-clase nth-child() y el número de índice correcto. Sin embargo, de este modo se generaría gran cantidad de código y sería imposible de usar en sitios web con generación dinámica de contenido. Una alternativa para obtener el mismo resultado con mayor eficacia sería tomar ventaja de las palabras clave odd y even, disponibles para esta pseudo-clase.

```
p:nth-child(odd){
    background: #999999;
}
p:nth-child(even){
    background: #cccccc;
}
```

Código 2-22

Sacar provecho de las palabras clave odd y even.

Ahora solo hay dos reglas para toda la lista (no importa cuán larga sea). Aunque se incorporen nuevas opciones o filas a la lista, no será necesario añadir reglas adicionales al archivo. Los estilos se asignarán automáticamente a cada elemento según su posición.

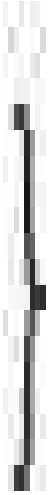
La palabra clave odd para la pseudo-clase nth-child( ) afecta a los elementos <p> que se encuentren anidados en otro elemento y tengan un índice o número de orden impar, y la palabra clave even afecta a aquellos que tienen un número de orden par.

Hay otras importantes pseudo-clases relacionadas con las que estamos estudiando, algunas de ellas de reciente incorporación, como first-child, last-child y only-child.

La pseudo-clase first-child hace referencia solo al primer elemento anidado, last-child hace referencia solo al último y only-child afecta a un elemento solo si es el único anidado en su elemento padre. Estas pseudo-clases en particular no requieren palabras clave o parámetros adicionales y se implementan como en el ejemplo de la página siguiente.

p:last-child{

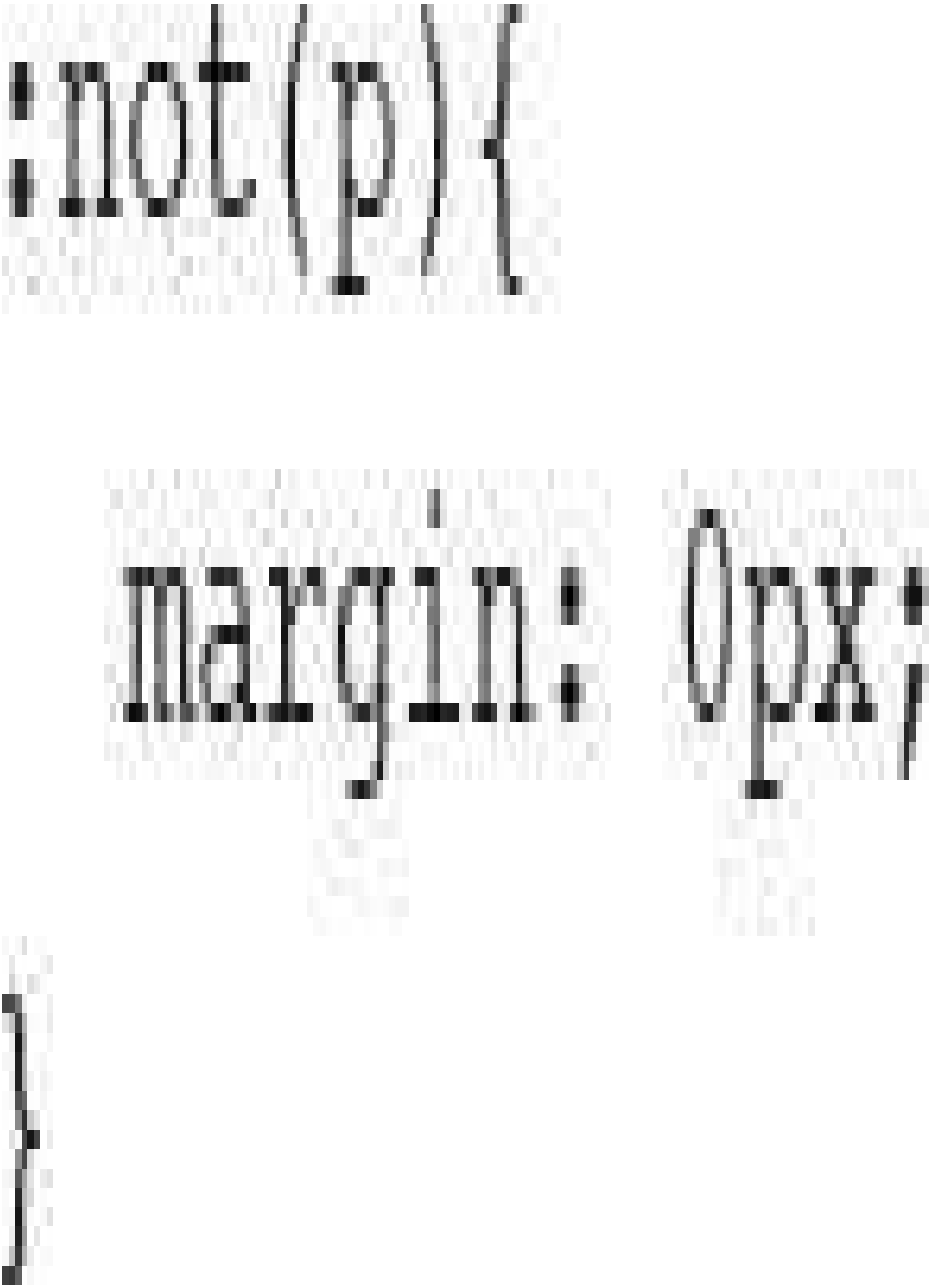
background: #ggggg;


}

Código 2-23

Usar: last-child para modificar solo el último elemento <p> de la lista.

Otra importante pseudo-clase es la de negación, llamada not().


:not(p){
margin: 0px;
}

Código 2-24

Aplicación de estilos a todos los elementos, excepto <p>.

La regla del Código 2-24 asigna un margen de 0 px a cada elemento en el documento excepto los elementos <p>. Los estilos de las reglas creadas con esta pseudo-clase se asignan a cada elemento, pero no a los incluidos entre paréntesis.

En lugar de la palabra clave del elemento, puede utilizar cualquier referencia que desee. En el Código 2-25 cada elemento se verá afectado, excepto aquellos cuyo valor sea mitexto2 en su atributo de clase.

```
:not(.mitexto2){
    margin: 0px;
}
```

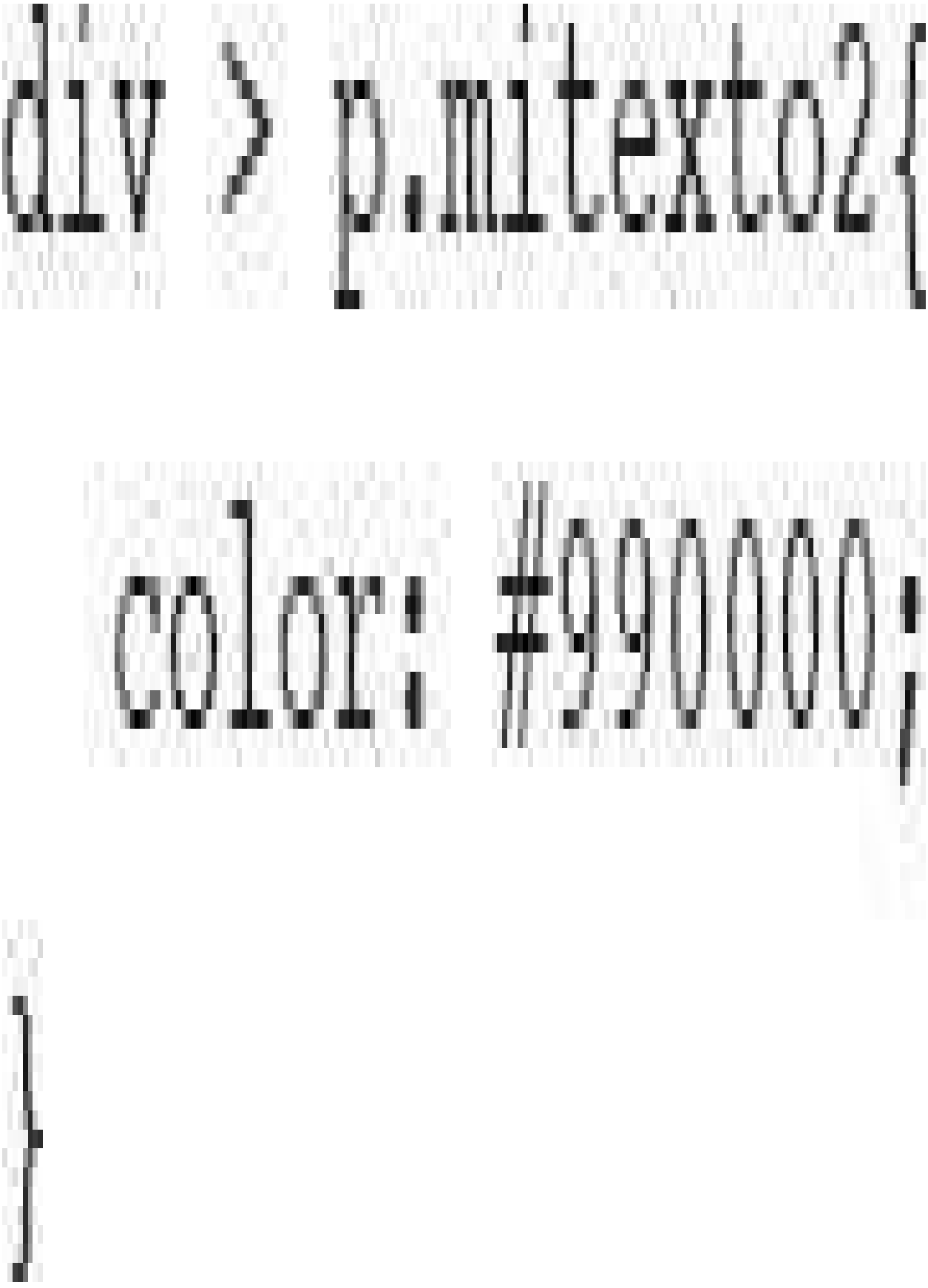
Código 2-25

Excepción utilizando el atributo class.

Al aplicar esta última regla en el documento HTML con el Código 2-18, el navegador asigna los estilos por defecto al elemento <p> identificado por el valor de clase mitexto2 y proporciona un margen de 0 px para el resto ya que el elemento <p> llamado mitexto2 no se ve afectado por la propiedad.

2.2.7 Selectores nuevos

Hay otros selectores que se agregaron o se consideraron parte de CSS3 y que son de utilidad a la hora de crear un diseño web. Estos selectores usan los símbolos >, + y ~ para especificar una relación entre los elementos.


div > p.mitexto2{
color: #990000;
}

Código 2-26:

Uso del selector >.

El selector > indica que el elemento afectado es el segundo elemento si éste está anidado en el primero. La regla del Código 2-26 modifica los elementos <p> si son hijos de un elemento . En este caso, la regla es más específica y está referida solo al elemento <p> de la clase mitexto2.

El selector siguiente se construye con el símbolo +, que hace referencia al segundo elemento cuando éste está precedido inmediatamente por el primer elemento. Ambos elementos deben compartir el mismo elemento padre.

```
p.mitexto2 + p{

    color: #990000;

}
```

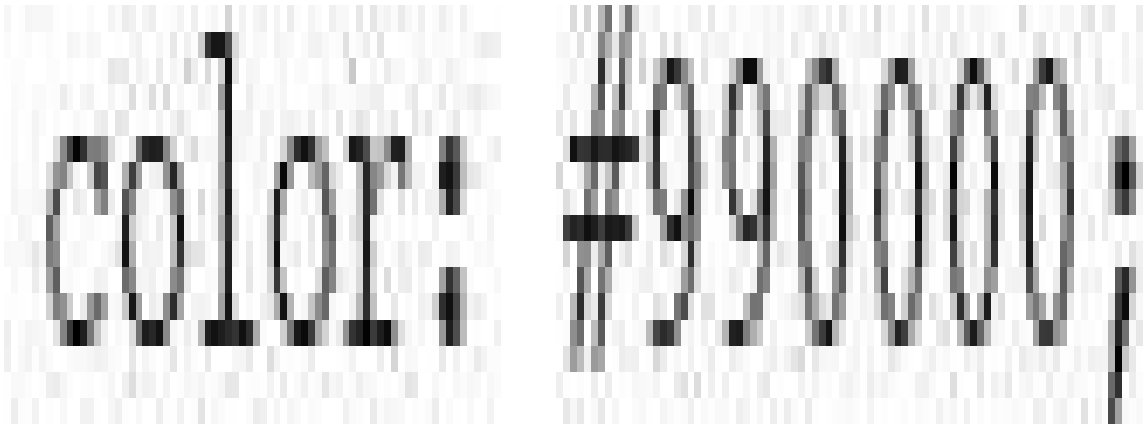
Código 2-27

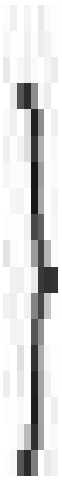
Uso del selector +.

La regla del Código 2-27 afecta a los elementos <p> que se encuentren después de otro elemento <p> identificado con la clase mitexto2. Si abre en el explorador el archivo HTML con el Código 2-18, el texto del tercer elemento <p> se mostrará de color rojo porque este elemento <p> en particular está inmediatamente después del elemento <p> identificado con la clase mitexto2.

El último selector se construye con el símbolo ~. Este selector es similar al anterior, pero el elemento afectado no tiene que ser inmediatamente anterior al primer elemento. Además, puede haber más de un elemento.

p.mitexto2 ~ p{


color: #990000;


}

Código 2-28

Uso del selector ~.

La regla en el Código 2-28 afecta al tercer y cuarto elemento <p> de nuestro ejemplo. El estilo se aplica a todos los elementos <p> que son hermanos y se encuentran después del elemento <p> identificado con la clase mitexto2, no importa si otros elementos están en medio.

Puede experimentar en el documento HTML que contiene el Código 2-18 mediante la inserción de un elemento   después del elemento <p> con la clase mitexto2 para verificar que solo los elementos <p> son modificados por esta regla.

2.3 Aplicar CSS a nuestro documento

Por defecto, todos los navegadores ordenan los elementos de un sitio web básicamente según su tipo: elementos de bloque o en línea. Esta clasificación se basa en la manera en la que los elementos se muestran en la pantalla.

Los elementos de bloque se colocan uno tras otro en la página.

Los elementos en línea se colocan de lado a lado, uno junto al otro en la misma línea, sin ningún salto de línea salvo que no haya suficiente espacio horizontal.

Casi todos los elementos estructurales de nuestro documento son tratados por defecto como un elemento de bloque. Eso significa que cada etiqueta HTML que representa una parte de la organización visual (por ejemplo, <section>, <nav>, <header>, <footer>,) se colocarán una debajo de la otra.

En el Capítulo 1 creamos un documento HTML para representar el diseño típico de una página web. Éste incluía barras horizontales y dos columnas en medio. Debido a la forma en la que los navegadores muestran estos elementos por defecto, el resultado obtenido en pantalla estaba muy lejos de lo esperado. Si abre el archivo HTML que contiene el Código 1-20 (Capítulo 1) en su navegador, podrá identificar inmediatamente las posiciones incorrectas de las columnas definidas como <section> y <aside>. Una está bajo la otra en lugar de estar una junto a la otra, debido a que cada bloque es mostrado por defecto con el mayor ancho posible, tan largo como la información que contiene y uno bajo el otro, como se muestra en la Figura 2-1.

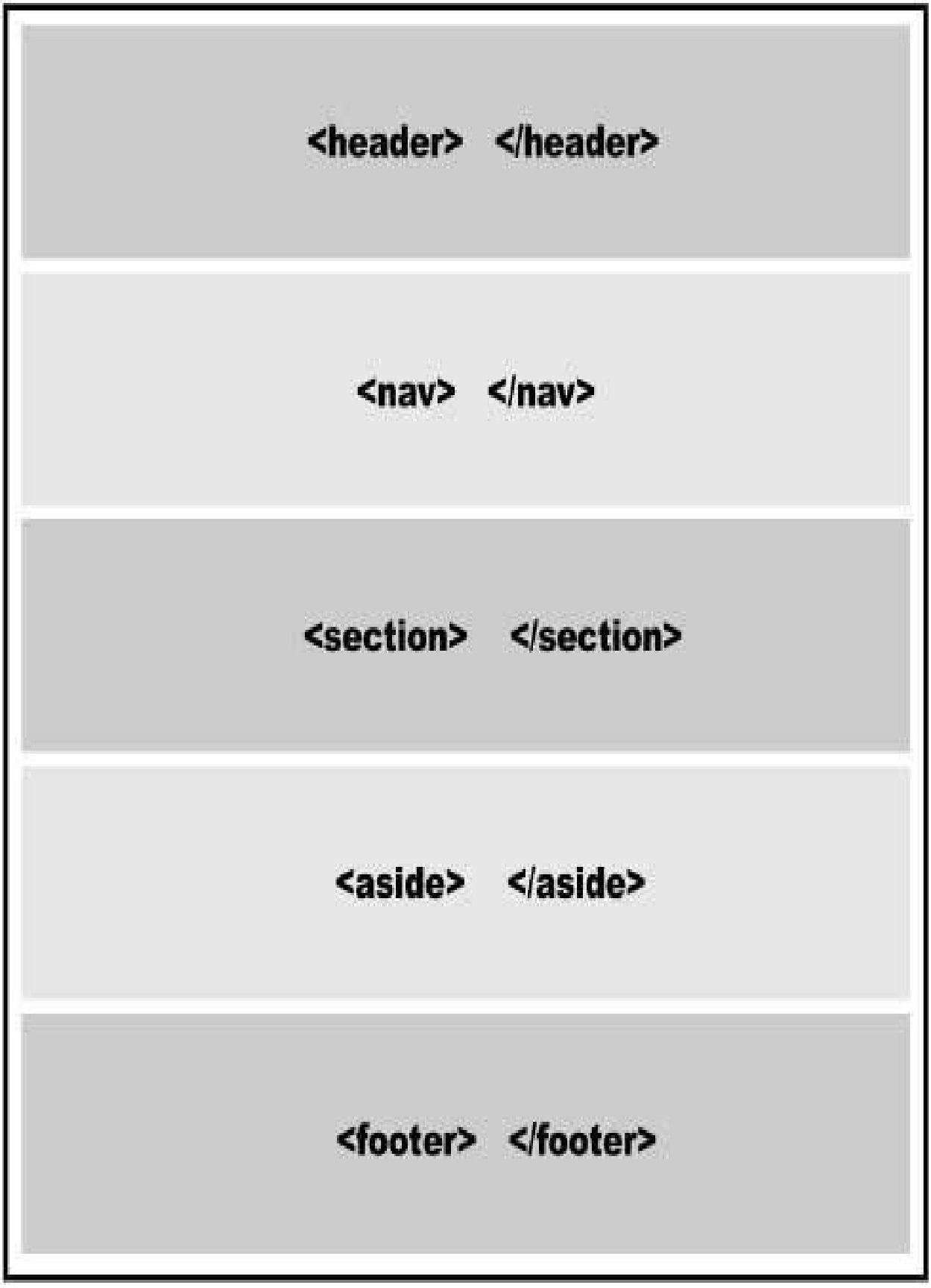

<header> </header>
<nav> </nav>
<section> </section>
<aside> </aside>
<footer> </footer>

Figura 2-1

Representación visual de un diseño de página con estilos predeterminados.

2.3.1 Modelos de caja

Para poder crear su propio diseño tendrá que entender primero la forma en la que los navegadores procesan el código HTML. Los navegadores consideran a cada elemento HTML como una caja y una página web es en realidad un grupo de cajas colocadas según ciertas reglas. Estas reglas son establecidas por los estilos proporcionados por los navegadores o por reglas CSS creadas por los diseñadores.

El lenguaje CSS tiene un conjunto predeterminado de propiedades para sobrescribir los estilos de los navegadores y lograr el diseño deseado. Estas propiedades no son específicas y tienen que ser combinadas para formar reglas que más adelante son utilizadas para agrupar cajas y obtener la distribución correcta. La combinación de estas normas se suele llamar modelo o sistema de diseño. La combinación de estas reglas constituye un modelo de caja.

Actualmente hay un modelo de caja estándar además de algunos otros experimentales. Este primer modelo, válido y ampliamente adoptado, es llamado el Modelo de caja tradicional y ha sido utilizado desde la primera versión de CSS. Aunque efectivamente ha demostrado ser un modelo eficaz, algunos modelos experimentales están tratando de resolver sus deficiencias. El más importante de ellos, que se considera parte de HTML5, es el nuevo Modelo de caja flexible, introducido con CSS3.

El Modelo de caja tradicional es actualmente soportado por todos los navegadores en el mercado y en realidad es un estándar para el diseño web. Por el contrario, el Modelo de caja flexible, incorporada en CSS3, está todavía en desarrollo aunque sus ventajas sobre el Modelo de caja tradicional podría convertirlo en el nuevo estándar; de ahí su importancia como objeto de estudio.

Ambos modelos pueden ser aplicados a una misma estructura HTML, pero la estructura tiene que estar preparada de forma correcta para ser modificada por el modelo seleccionado.

2.4 Modelo de caja tradicional

En un comienzo el diseño web se hacía a partir de tablas: elementos que casi sin querer se convirtieron en herramientas que permitían organizar cajas de contenido en pantalla. Esto puede considerarse el primer modelo de caja de la Web y entonces las cajas eran creadas expandiendo celdas y combinando filas, columnas y tablas enteras, una al lado de la otra o incluso anidándolas. Cuando los sitios web se hicieron más grandes y complejos, esta práctica reveló graves problemas debido al tamaño de los archivos y al mantenimiento de código que exigían.

Importante

El Modelo de caja tradicional presentado a continuación no es una incorporación de HTML5. Este modelo está disponible desde la primera versión de CSS y es probable que ya sepa cómo utilizarlo. Si este es el caso, no dude pasar a la siguiente sección de este capítulo.

Estos problemas iniciales abrieron camino a lo que ahora parece natural: la división entre estructura y presentación. El uso de etiquetas y de estilos CSS hizo que fuera posible sustituir la función de las tablas y separar la estructura HTML de la presentación. Efectivamente, con elementos y CSS podemos crear cajas en la pantalla, colocarlas en un lado u otro y darles un tamaño específico, borde, color, etc. CSS proporciona propiedades específicas que permiten organizar las cajas de acuerdo a las necesidades y estas propiedades han sido lo suficientemente poderosas como para crear un modelo de cajas que se ha desarrollado hasta dar lugar a lo que hoy se conoce como el

Modelo de caja tradicional.

Debido a algunas deficiencias de este modelo, las tablas se siguieron usando por un tiempo pero, bajo la influencia del éxito de Ajax y una gran cantidad de nuevas aplicaciones interactivas, los elementos y los estilos CSS se convirtieron poco a poco en el nuevo estándar. Finalmente, el modelo de caja tradicional fue adoptado a gran escala.

2.4.1 Documento HTML

En el Capítulo 1 construimos un documento HTML. Este documento tiene los elementos necesarios para dotar de estructura y organización a sus contenidos, pero es necesario añadir algunos elementos para prepararlo para la aplicación del Modelo de caja tradicional de CSS.

El Modelo de caja tradicional exige que las cajas estén agrupadas para poder ordenarlas de forma horizontal. Como todo el contenido del cuerpo de la página está compuesto por un conjunto de cajas a las que hay que dar una alineación y un tamaño específicos, es necesario que todas estas cajas estén agrupadas por un elemento .

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <meta name="description" content="Ejemplo de HTML">
  <meta name="keywords" content="HTML5, CSS3, JavaScript">
  <title>Éste texto es el título del documento</title>
  <link rel="stylesheet" href="misestilos.css">
</head>
<body>
<div id="caja_principal">
  <header id="cabecera">
    <h1>Título principal del sitio</h1>
  </header>
  <nav id="menú">
    <ul>
      <li>Inicio</li>
      <li>Imágenes</li>
      <li>Vídeos</li>
      <li>Contacto</li>
    </ul>
  </nav>
  <section id="área_principal">
    <article>
      <header>
        <hgroup>
          <h1>Título de la primera entrada</h1>
          <h2>Subtítulo de la primera entrada</h2>
        </hgroup>
        <time datetime="2013-01-02" pubdate>Publicado el 12-01-2013<
                    time>
      </header>
      Éste es el texto de la primera entrada.
      <figure>
        <img src="http://minkbooks.com/content/myimage.jpg">
        <figcaption>
          Ésta es la imagen de mi primera entrada
        </figcaption>
      </figure>
      <footer>
        <p>Comentarios (0)</p>
      </footer>
    </article>
    <article>
      <header>
        <hgroup>
          <h1>Título de la segunda entrada</h1>
          <h2>Subítulo de la segunda entrada</h2>
        </hgroup>
        <time datetime="2013-01-15" pubdate>Publicado el 15-01
                    2013</time>
      </header>
      Éste es el texto de la segunda entrada.
      <footer>
        <p>Comentarios (0)</p>
      </footer>
    </article>
  </section>
  <aside id="barra_lateral">
    <blockquote>Cita del Artículo 1</blockquote>
    <blockquote>Cita del Artículo 1</blockquote>
  </aside>
  <footer id="pie">
    Copyright &copy 2013
  </footer>
</div>
</body>
</html>
```

Codigo 2-29

Preparación del documento HTML5 para un estilo CSS.

El Código 2-29 proporciona un nuevo documento HTML listo aplicarle los estilos CSS. Hay dos cambios importantes respecto al Código 1-20 (Capítulo 1). En este nuevo código varias etiquetas han sido identificadas con los atributos id y class. Esto significa que ahora podemos hacer referencia a elementos muy específicos usando el valor de su atributo id, o podemos modificar varios elementos al mismo tiempo usando el valor de su atributo class. La segunda modificación importante respecto al documento con el que trabajamos antes, es el uso del elemento mencionado anteriormente. El elemento fue identificado con el atributo id="caja_principal" y se cierra al final del cuerpo con la etiqueta de cierre . Precisamente este elemento nos permitirá aplicar el modelo de caja al contenido del cuerpo y designar su posición horizontal.

Hágalo usted mismo

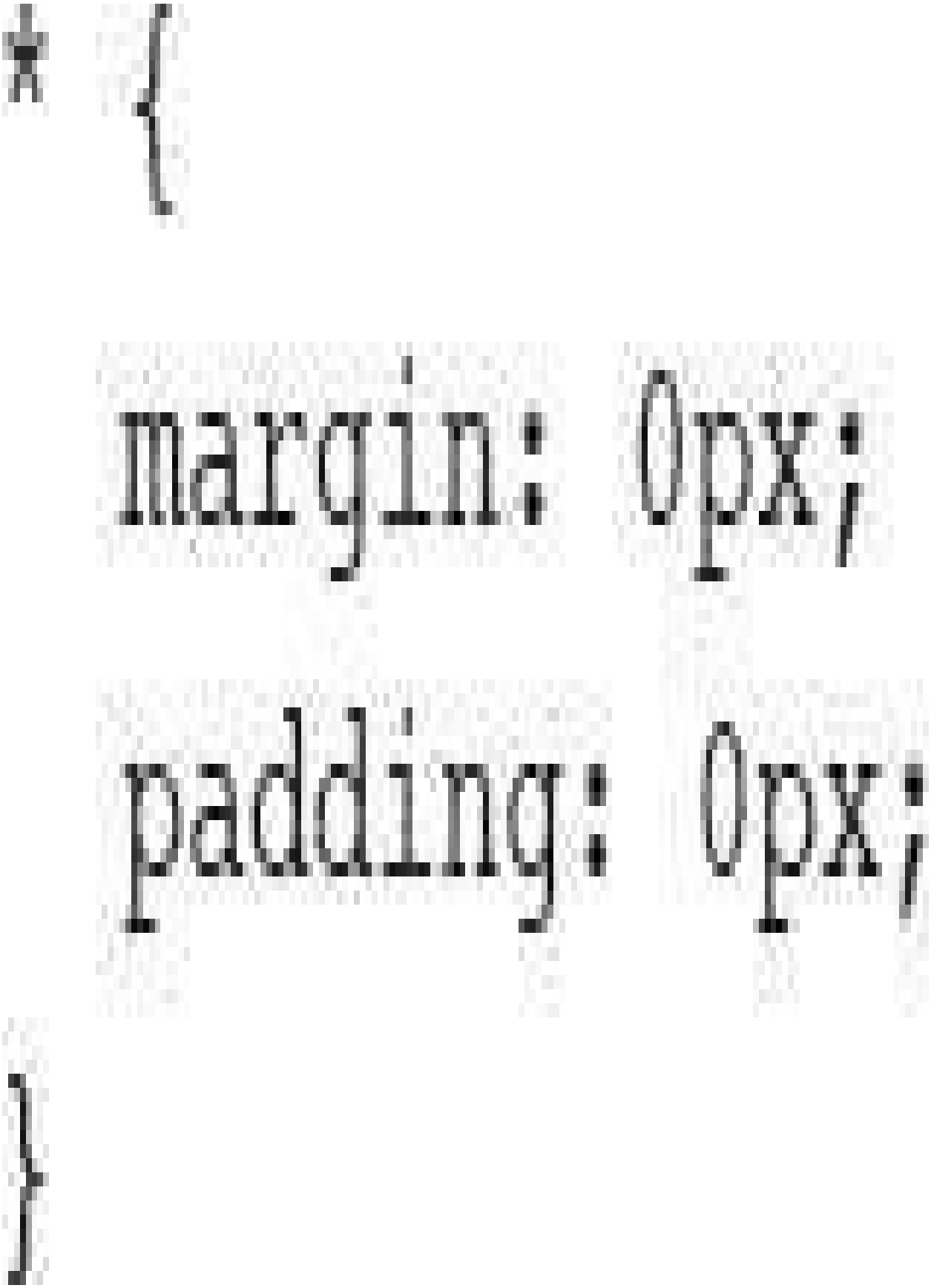
Compare el Código 1-20 (Capítulo 1) con el Código 2-29 para comprobar la ubicación de la etiquetas de apertura y cierre del elemento añadido como contenedor. Compruebe también cuáles elementos del segundo código están identificados con el atributo id y cuáles con el atributo class. Confirme que los valores de los atributos id son únicos para cada etiqueta. Reemplace el contenido del archivo HTML creado anteriormente por el Código 2-29 antes de aplicar las siguientes reglas CSS.

Una vez el documento HTML está listo, ha llegado la hora de crear la hoja de estilos de nuestro sitio de ejemplo.

2.4.2 Selector universal (*)

El selector universal * asigna la misma regla a cada elemento del cuerpo del documento y es útil para establecer algunas reglas básicas.

Por lo general, para la mayoría de los elementos es necesario personalizar los márgenes, o simplemente mantener unos márgenes mínimos. Algunos de los elementos tienen por defecto márgenes mayores que cero, que a menudo resultan excesivos. A medida que se avanza en la creación del diseño, sucede los márgenes de la mayor parte de los elementos tienen que ser puestos a cero. Para evitar reglas repetitivas, puede utilizar el selector universal.


*
{
margin: 0px;
padding: 0px;
}

Codigo 2-30

Introducción de una regla general de CSS.

La primera regla CSS que aparece en el Código 2-30, establece un margen de 0 px para cada elemento. De este modo solo será necesario modificar los márgenes para aquellos elementos que queramos que tengan un margen mayor.

Hágalo usted mismo

Ahora deberá escribir todas las reglas CSS necesarias en el archivo de hoja de estilo. El archivo fue incluido en la cabecera del código HTML con la etiqueta <link>, así que todo lo que tendrá que hacer es crear con su editor de texto un nuevo archivo llamado misestilos.css e insertar todas las reglas presentadas a partir del Código 2-30.

2.4.3 Títulos

Usamos los elementos <h1> y <h2> en el documento para declarar los títulos y subtítulos de las diferentes secciones. El elemento <h1>, por ejemplo, fue aplicado varias veces, no solo para el título principal de todo el documento sino también para algunas secciones internas. Ahora hay que aplicarles estilos apropiados:

```
h1 {
    font: bold 20px verdana, sans-serif;
}
h2 {
    font: bold 14px verdana, sans-serif;
}
```

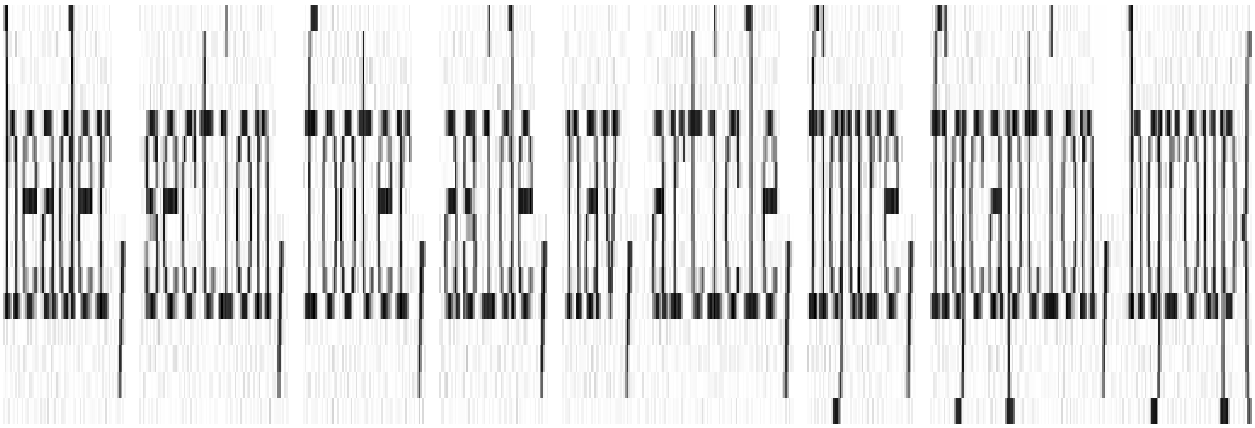
Código 2-31

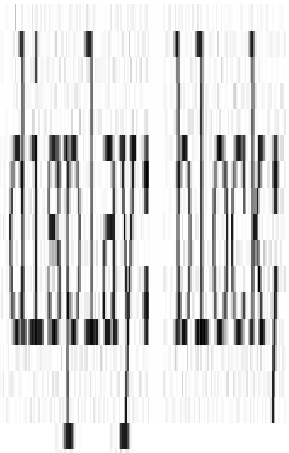
Adición de estilos para los elementos <h1> y <h2>.

En la regla anterior, la propiedad font cambia la familia, el tipo, el tamaño y la fuente de todos los textos comprendidos dentro de los elementos <h1> y <h2>.

2.4.4 Declaración de nuevos elementos HTML5

Otra regla básica que debe ser declarada al principio de un documento es la definición por defecto de los elementos estructurales de HTML5. Algunos navegadores no reconocen estos elementos o los tratan como elementos en línea. Por tanto siempre es necesario declararlos como elementos de bloque para asegurar que serán tratados como elementos y que por lo tanto será posible construir el modelo de caja.





Código 2-32

Definición de los elementos de HTML5 como elementos de bloque.

Ahora los elementos afectados por la regla del Código 2-32 se colocarán uno sobre el otro a menos que se especifique lo contrario más adelante.

Conceptos básicos

La propiedad display asigna un tipo diferente al elemento afectado. Si quiere convertir un elemento en línea en un elemento de bloque, lo único que tiene que hacer es utilizar la declaración display: block. Hay varios posibles valores para esta propiedad, como block, inline, inline-block, etc. Para obtener una lista completa, por favor visite nuestro sitio web y siga los enlaces propuestos para este capítulo.

2.4.5 Centrar el cuerpo

El primer elemento del modelo de caja es siempre <body> y usualmente su contenido tiene estar de forma horizontal. Para obtener un diseño consistente en diferentes configuraciones de pantalla se debe indicar su tamaño o su tamaño máximo.

body {
text-align: center;
}

Codigo 2-33

Centrado del cuerpo.

De forma predeterminada, la etiqueta <body> tiene un valor de ancho del 100%. Esto significa que el cuerpo va a ocupar todo el ancho de la ventana del navegador. Para centrar la página en la ventana, es necesario centrar el contenido en el interior del cuerpo. Con la regla añadida en el Código 2-33, todo lo que esté dentro del elemento <body> se centra, centrando por tanto toda la página web.

2.4.6 Creación de la caja principal

Hay que especificar un tamaño para el contenido del cuerpo. Recordará que en el Código 2-29 añadimos un elemento id=“caja_principal” para agrupar el contenido del elemento <body>, que ahora está contenido en su interior. El tamaño de esta caja determinará el tamaño máximo del resto de los elementos.

```
#caja_principal {
  width: 960px;
  margin: 15px auto;
  text-align: left;
}
```

Código 2-34

Definición de las propiedades de la caja principal.

La regla del Código 2-34 es la primera que hace referencia a un elemento mediante el valor de su atributo id. El carácter # indica al navegador que el elemento afectado por estos estilos tiene el atributo id con el valor caja_principal.

Ésta regla establece tres estilos para el cuadro principal. El primero establece un ancho fijo de 960 px para la caja. (Los valores habituales para un sitio web actualmente están entre 960 y 980 px de ancho). El segundo estilo es parte del Modelo de caja tradicional. En la regla anterior (Código 2-33), usamos el estilo text-align: center para establecer que el contenido del cuerpo se centre, pero esto solo afecta al contenido en línea, como pueden ser textos o imágenes. Para los elementos en bloque como , es necesario establecer un valor para sus márgenes que los ajusten automáticamente al tamaño del elemento padre. En el Código 2-34, el estilo margin: 15px auto asigna 15 px para el margen superior e inferior del elemento al que hace referencia y establece el tamaño de los márgenes izquierdo y derecho como automático. De este modo tendremos un espacio de 15 px en las partes superior e inferior del cuerpo y el espacio a la izquierda y la derecha será calculado automáticamente de acuerdo con el tamaño del cuerpo y el tamaño del elemento , centrando efectivamente el contenido en la pantalla. Ahora la página web está centrada y con un tamaño fijo de 960 px.

Con el siguiente paso evitaremos un problema que se produce en algunos navegadores. La propiedad text-align es hereditaria. Esto significa que no solo la caja principal estará centrada, sino que también lo estarán todos los elementos que contiene: el estilo se transferirá a cada elemento anidado dentro del elemento

<body>. Por tanto es necesario volver a cambiar este estilo a su valor por defecto para el resto del documento. El tercero y último estilo de la regla del Código 2-34 se encargará de esto. De este modo el contenido del cuerpo estará centrado pero el contenido de la caja principal (el elemento) estará de nuevo alineado a la izquierda, y de este modo, el resto del código heredará este estilo y lo utilizará de forma predeterminada.

Hágalo usted mismo

Si aún no lo ha hecho, copie cada regla encontrada hasta este punto en un archivo vacío con el nombre misestilos.css. Este archivo debe ser colocado en la misma carpeta o directorio que el archivo HTML que contiene el Código 2-29. Deberá tener por lo menos dos archivos, uno con el código HTML y otro llamado mystyles.css con todos los estilos CSS que estudiamos desde el Código 2-30. Abra el archivo HTML en el navegador para ver en pantalla el contenido creado.

2.4.7 La cabecera

Continuemos con el resto de los elementos estructurales. A continuación de la etiqueta de apertura se encuentra el primer elemento estructural de HTML5: <header>. Este elemento contiene el título principal de la página web y se encuentra en la parte superior de la pantalla. Hemos identificado el elemento <header> en el código con el atributo id=“cabecera”.

Por defecto, cada elemento en bloque, tal como sucedía con el cuerpo, tiene un ancho del 100%. Esto significa que el elemento ocupa todo el espacio horizontal disponible. En el caso del cuerpo, como ya se ha dicho, este espacio es el ancho de la ventana del navegador pero para el resto de los elementos el espacio máximo se determina por el tamaño de su elemento padre. En nuestro ejemplo, el espacio máximo de los elementos dentro de la caja principal será 960 px, ya que están anidados dentro de la caja principal y éste es el ancho establecido para este elemento.

```
#cabecera {
    background: #FFFBB9;
    border: 1px solid #999999;
    padding: 20px;
}
```

Código 2-35

Adición de estilos de <header>.

Dado que el elemento <header> ocupará todo el espacio horizontal disponible en la caja principal, y será tratado como un elemento de bloque y ubicado en la parte superior de la página, el siguiente paso es asignar estilos que nos permitan reconocerlo cuando se muestre en la pantalla. En la regla mostrada en el Código 2-35 le damos al elemento <header> un fondo amarillo, un borde sólido de 1 píxel y un margen interior de 20 px determinado por la propiedad padding.

2.4.8 Barra de navegación

Tras el elemento <header>, el siguiente elemento estructural es <nav>, que tiene el propósito de facilitar la navegación. Los vínculos agrupados dentro de este elemento representarán el menú del sitio web. Este menú será solo una barra ubicada debajo de la cabecera. Tal como sucedía con el elemento <header>, la mayoría de los estilos que necesitamos para posicionar el elemento <nav> ya han sido asignados: <nav> es un elemento de bloque, por tanto, se ubicará debajo del elemento anterior; el ancho predeterminado es 100%, por lo que será tan amplio como su padre (la caja principal) y, también por defecto, será tan alto como su contenido y los márgenes establecidos. De esta forma, solo tenemos que hacerlo más atractivo y esto lo haremos añadiendo un fondo gris y un pequeño margen interno para separar las opciones del menú del borde de éste.

```
#menu {
  background: #CCCCCC;
  padding: 5px 15px;
}
#menu li {
  display: inline-block;
  list-style: none;
  padding: 5px;
  font: bold 14px verdana, sans-serif;
}
```

Código 2-36

Adición de estilos de <nav>.

En el Código 2-36, la primera regla hace referencia al elemento <nav> a través de su id, luego cambia su fondo y añade márgenes internos de 5 px y 15 px usando la propiedad padding. Por otra parte, dentro de la barra de navegación hay una lista creada con los elementos <ul> y <li>. De forma predeterminada, los elementos de una lista se sitúan uno bajo el otro. Para modificar esto y poner todas las opciones del menú en línea una junto a la otra, creamos referencias a los elementos <li> ubicados dentro de este elemento <nav> en particular usando el selector #menu li. Finalmente asignamos el estilo display: inline-block para cambiar su tipo. A diferencia de los elementos block, los elementos inline-block, estandarizado en CSS3, no generan saltos de línea pero pueden ser tratados como bloques y tener ancho fijo. Si no se establece un ancho, éste parámetro establece el tamaño del elemento de acuerdo con el tamaño de su contenido.

Conceptos básicos

La propiedad list-style se refiere a los pequeños gráficos que se ubican delante de los elementos de una lista (normalmente llamados viñetas). En nuestro ejemplo, hemos asignado el valor none para esta propiedad para eliminar las viñetas. Sin embargo, hay muchos valores disponibles, tales como square, circle, decimal, etc. La propiedad permite declarar no solo el tipo de gráfico sino también la posición (inside o outside) y una imagen personalizada (por ejemplo, list-style: url (’mybullet.jpg’);).

2.4.9 Área principal y Barra lateral

Los siguientes elementos estructurales en el código son dos cajas colocadas horizontalmente. El Modelo de caja tradicional permite especificar la posición de cada caja. Con la propiedad float es posible colocarlas en el lado derecho o izquierdo de la pantalla de acuerdo con las propias necesidades. Los elementos que utilizamos en el documento HTML para ello son <section> y <aside>, cada uno identificado con el atributo id y los valores area_principal y barra_lateral.

```
#area_principal {
   float: left;
   width: 660px;
   margin: 20px;
}
#barra_lateral {
   float: left;
   width: 220px;
   margin: 20px 0px;
   padding: 20px;
   background: #CCCCCC;
}
```

Código 2-37

Creación de dos columnas con la propiedad float.

La propiedad CSS float es una de las ampliamente utilizadas para aplicar el Modelo de caja tradicional y hace que el elemento flote a un lado u otro en el espacio disponible. Los elementos modificados por esta propiedad actúan como elementos de bloque, pero están dispuestos de acuerdo con el valor de esta propiedad y no con el flujo normal del documento: se mueven a la izquierda o a la derecha del área disponible, tan lejos como sea posible. Con el Código 2-37 declaramos la posición de las dos cajas y sus tamaños, y generamos así dos columnas. La propiedad float mueve la caja al espacio disponible en el lado indicado por su valor, width asigna un tamaño horizontal, y margin, desde luego, establece el margen del elemento.

Una vez aplicados estos estilos, el contenido del elemento <section> se alineará al lado izquierdo de la pantalla, con un tamaño de 660 px más 40 px de márgenes, de manera que ocupará un espacio total de 700 px de ancho.

La propiedad float del elemento <aside> también tiene el valor left. Esto significa que la caja generada se moverá hacia el espacio disponible a su izquierda. Dado que la caja anterior creada con el elemento <section> también fue movida a la izquierda de la pantalla, ahora el espacio disponible será el restante. La nueva caja estará en la misma línea que la primera pero a su derecha, ocupando el resto del espacio y dando lugar a una segunda columna. Podríamos haber utilizado el valor right para la propiedad float de <aside> porque solo tenemos dos columnas, una a la izquierda y una a la derecha de la pantalla, pero el uso del valor left deja abierta la posibilidad añadir posteriormente nuevas columnas al diseño.

También declaramos un tamaño de 220 px para esta segunda caja, agregamos un fondo de color gris y establecimos un margen interior de 20 px. Como resultado, el tamaño de esta caja horizontal será 22 px, más 40 px añadidos por la propiedad padding; por otra parte, los márgenes laterales se han fijado en 0 px.

Conceptos básicos

El tamaño de un elemento y sus márgenes se indican para obtener el valor real. Si tenemos un elemento de 200 px de ancho y un margen de 10 px a cada lado, el ancho real del elemento será 220 px. Los 20 px del total de márgenes se añaden a los 200 px del elemento, y el valor final se representa en la pantalla. Lo mismo sucede con las propiedades padding y border. Cada vez que añade un borde a un elemento o crea un espacio entre el contenido y el borde con la propiedad padding, sus valores se añaden al ancho total del elemento para obtener el valor real mostrado en pantalla. El valor real se calcula mediante la fórmula: size+margin+padding+borders.

Hágalo usted mismo

Revise el Código 2-29. Consulte todas las reglas CSS que hemos creado hasta el momento y busque en el documento HTML el elemento correspondiente a cada una de esas reglas. Siga las referencias, las palabras clave (como h1) y los atributos id (como cabecera), para entender cómo funcionan las referencias y cómo son asignados los estilos a cada elemento.

2.4.10 Pie de página

Para finalizar la aplicación del modelo de caja tradicional, es necesario aplicar otra propiedad de CSS al elemento <footer> que le devuelva el flujo normal al documento y permita posicionar el pie bajo el último elemento en lugar de a su lado.

```
#pie {
    clear: both;
    text-align: center;
    padding: 20px;
    border-top: 2px solid #999999;
}
```

Código 2-38

Aplicar estilo al elemento <footer> y recuperar el flujo normal.

El Código 2-38 declara un borde de 2 px en la parte superior del elemento <footer>, un margen interno (o relleno) de 20 px y centra el texto dentro del elemento. Además, se ha restaurado el flujo normal del documento con la propiedad clear, que simplemente limpia el área ocupada por el elemento para impedir que se ubique en una posición adyacente a una caja flotante. El valor habitual es both, que indica que ambos lados serán restaurados y el elemento recuperará el flujo habitual o, lo que es lo mismo, no será flotante. Para un elemento en bloque, esto significa que va a ser posicionado debajo del último elemento, en una nueva línea.

La propiedad clear también empuja los elementos verticalmente, haciendo que las cajas flotantes ocupen una superficie real en la pantalla. Sin esta propiedad, el navegador muestra el documento como si los elementos flotantes no existieran y las cajas se superpusieran. Al posicionar las cajas una junto a la otra según el Modelo de caja tradicional, se hace indispensable el uso del estilo clear: both para poder colocar otras cajas debajo. La Figura 2-2 muestra una representación visual de este modelo con los estilos CSS básicos necesarios para crear el diseño.

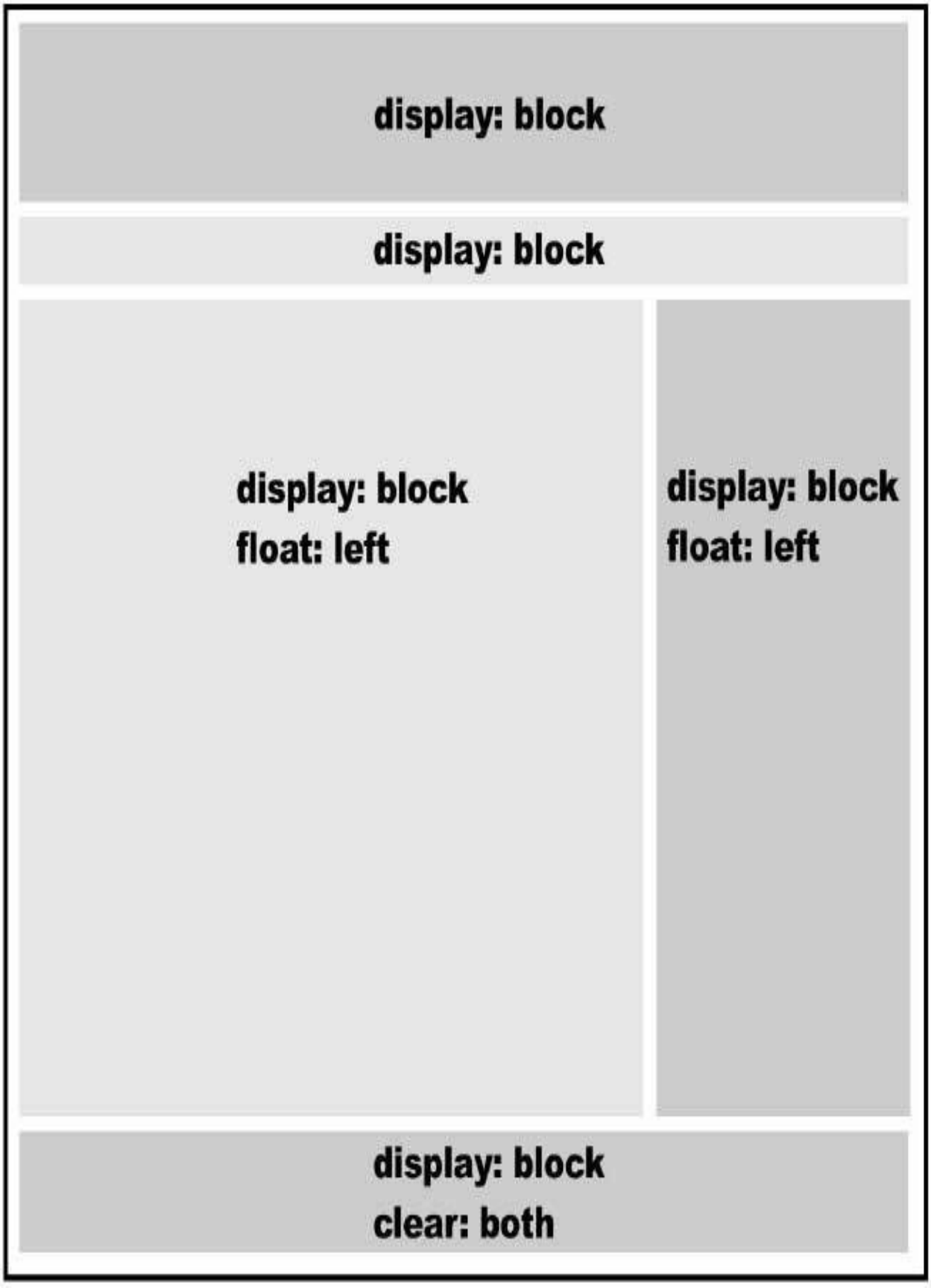

display: block
display: block
display: block
float: left
display: block
float: left
display: block
clear: both

Figura 2-2

Representación visual del modelo de caja tradicional.

Los valores left y right para la propiedad float no indican necesariamente que la caja estará a la izquierda o la derecha de la ventana. Estos valores hacen que sea flotante específicamente el lado indicado del elemento y rompen de este modo el flujo normal del documento. Si el valor es left, por ejemplo, el navegador intentará ubicar el elemento en el lado izquierdo del espacio disponible. Si no hay espacio disponible al lado del elemento anterior, el nuevo elemento se ubicará a su derecha, porque la propiedad float le ha sido aplicada al lado izquierdo del elemento.

Tenga esto en cuenta cuando quiera crear un diseño con varias columnas. Deberá aplicar a cada columna la propiedad float con un valor left para garantizar que cada una sea colocada al lado de la anterior en el orden correcto, flotando hacia la izquierda hasta que alguna sea bloqueada por otra columna o por el borde del contenedor. De este modo las cajas se ubicarán alineadas una junto a la otra, creando un conjunto de columnas en pantalla.

2.4.11 Toques finales

El último paso es trabajar en el diseño de los contenidos. Hay algunos elementos de HTML5 adicionales que pueden ser útiles para este propósito:

```
article {
  background: #FFFBCC;
  border: 1px solid #999999;
  padding: 20px;
  margin-bottom: 15px;
}
article footer {
  text-align: right;
}
time {
  color: #999999;
}
figcaption {
  font: italic 14px verdana, sans-serif;
}
```

Código 2-39

Añadir los toques finales para el diseño básico.

La primera regla del Código 2-39 hace referencia a todos los elementos <article> y les aplica algunos estilos: color de fondo, borde sólido de 1 px, relleno y margen inferior. El margen inferior de 15 px separa a cada artículo del siguiente.

Cada artículo tiene también un elemento <footer> que muestra el número de comentarios recibidos. Para hacer referencia a un elemento <footer> dentro de un elemento <article>, hemos utilizado el selector article footer que indica que cada elemento <footer> que se encuentre dentro de un elemento <article> se verá modificado por la regla. Esta técnica de referencia se aplica en este caso para alinear a la derecha el texto del pie de cada artículo.

Al final del Código 2-39 se modifica el color de cada elemento <time> y se distigue la leyenda de la imagen (figcaption) del resto de textos del artículo usando un tipo de letra diferente.

Título principal del sitio

Inicio Imágenes Vídeos Contacto

Título de la primera entrada

Subtítulo de la primera entrada

Publicado el 12-01-2013

Éste es el texto de la primera entrada.

Ésta es la imagen de mi primera entrada

Comentarios (0)

Título de la segunda entrada

Subítulo de la segunda entrada

Publicado el 15-01-2013

Éste es el texto de la segunda entrada.

Comentarios (0)

Cita del Artículo 1
Cita del Artículo 1

Copyright © 2013-2014

Figura 2-3

El modelo de caja tradicional aplicado al documento.

Hágalo usted mismo

Si no lo ha hecho aún, copie todas las reglas CSS que figuran en este capítulo desde Código 2-30, uno tras otro, en el interior del archivo misestilos.css y a continuación abra el archivo HTML con el documento creado en el Código 2-29 en su navegador. El resultado debe ser similar a la Figura 2-3.

2.4.12 box-sizing

Otra propiedad incorporada en CSS3, que guarda relación con la estructura y el Modelo de caja tradicional, es la llamada box-sizing, que permite cambiar la forma en la que el tamaño de un elemento es calculado y obliga a los exploradores a incluir el relleno y el borde con su valor original.

Recordará que cada vez que calcula el área total ocupada por un elemento, el navegador utiliza la siguiente fórmula: size+margin+padding+borders. Por tanto, si la propiedad width tiene un valor de 100 px, margin tiene un valor de 20 px, padding tiene un valor de 10 px y border tiene un valor de 1 px, el área horizontal total ocupada por el elemento será de 100 + 40 + 20 + 2 = 162 px.

Observe que hemos tenido que duplicar los valores de las propiedades margin, padding y border para tomar en cuenta los valores que utilizados a la izquierda ya la derecha de la caja. Esto significa que cada vez que se declara el tamaño de un elemento con la propiedad width, es necesario tomar en cuenta que el verdadero espacio necesario para colocar el elemento suele ser mayor.

En ocasiones puede ser recomendable forzar al navegador a incluir los valores de padding y border en el valor la propiedad width, de manera que la nueva fórmula será igual size+margin.

```
div {
  width: 100px;
  margin: 20px;
  padding: 10px;
  border: 1px solid #000000;

  -moz-box-sizing: border-box;
  -webkit-box-sizing: border-box;
  box-sizing: border-box;
}
```

Código 2-40

Incluye relleno y borde en el tamaño del elemento.

La propiedad box-sizing puede tomar uno de tres valores. De forma predeterminada se establece el valor content-box, que indica que el navegador agregará los valores de padding y border al tamaño especificado por la propiedad width. Por otra parte, el valor padding-box incluirá el valor de padding dentro del elemento y border-box incluirá también el borde de la caja.

El Código 2-40 muestra la aplicación de esta propiedad a un elemento . Se trata solo de un ejemplo que no vamos a utilizar en nuestro documento, pero podría ser útil para algunos diseñadores dependiendo de qué tan familiarizados están con los métodos tradicionales de cálculo utilizados en las versiones anteriores de CSS.

Importante

Debido a que las propiedades CSS que estamos estudiando son aún experimentales, muchas deben ser declaradas con un prefijo específico de acuerdo con el motor de procesamiento. En el futuro estaremos en condiciones de declarar solo box-sizing: border-box, pero mientras no haya terminado esta fase experimental tenemos que utilizar una regla similar a la del Código 2-40.

Los prefijos para los navegadores más comunes son:

- **-moz-: para Mozilla Firefox.**

- **-webkit-: para Safari y Google Chrome.**

- **-o-: para Opera.**

- **-khtml-: para Konqueror.**

- **-ms-: para Internet Explorer.**

2.5 Modelo de caja flexible

El propósito principal de un modelo de caja es hacer que sea posible dividir el espacio de la ventana en varias cajas y, así, crear las filas y columnas que formen un diseño web regular. Sin embargo, el Modelo de caja tradicional, aplicado desde la primera versión de CSS y ampliamente utilizado hoy en día, fracasa en ese sentido. Por ejemplo, con este modelo no es posible definir de manera eficiente cómo se distribuyen las cajas y especificar su tamaño horizontal y vertical sin usar trucos y reglas intrincadas programadas por un tipo brillante en algún lugar del mundo.

Importante

Aunque el Modelo de caja flexible tiene ventajas sobre el modelo anterior, todavía es experimental y de momento no puede ser adoptado por navegadores y desarrolladores. En la actualidad existen dos especificaciones disponibles y una de ellas de momento solo es compatible con Google Chrome. Por eso tratamos la utilización del Modelo de caja tradicional en profundidad. Debe tener en cuenta estas cuestiones antes de escoger un modelo u otro. Al final del libro explicaremos cómo desarrollar diferentes versiones de una página web para todo tipo de navegadores, compatibles o no con HTML5.

Las dificultades para crear efectos simples como ampliar varias columnas de acuerdo con el espacio disponible, centrar verticalmente el contenido o ampliar una columna de arriba a abajo independientemente de su contenido, hizo que los desarrolladores comenzaran a pensar en cómo aplicar nuevos modelos a sus documentos y el Modelo de caja flexible es el que ha tenido más éxito de

momento.

El Modelo de caja flexible resuelve los problemas del Modelo de caja tradicional de una manera elegante. En este nuevo modelo las cajas finalmente representan las filas y columnas virtuales que diseñadores y usuarios realmente ven en pantalla, y que son, de hecho, lo que les preocupa a los primeros. Con este modelo se consigue un control total sobre el diseño, la posición y el tamaño de las cajas, la distribución de unas cajas dentro de otras, y la forma en la que éstas usan y comparten el espacio disponible. Finalmente el código satisface las necesidades de los diseñadores.

En esta sección del capítulo veremos cómo funciona el modelo de caja flexible, sus ventajas y cómo puede ser aplicado a un documento HTML.

2.5.1 Contenedor flexible

El motivo principal para la creación del Modelo de caja flexible fue la necesidad de distribuir los elementos del documento en la ventana. Originalmente los elementos tenían que ser reducidos o ampliados de acuerdo con el espacio disponible y para conocer el espacio disponible se hacía necesario saber el tamaño exacto del contenedor. Esto condujo a la definición de los contenedores flexibles.

Un contenedor flexible es un elemento que permite que su contenido se adapte. Algunas de las características importantes de este modelo, como la orientación vertical y horizontal, se declaran en los contenedores. Los elementos flexibles deben tener un elemento padre común dentro del cual puedan ser organizados, es decir, en este modelo cada conjunto de cajas tiene que estar anidado dentro de otra caja.

2.5.2 Documento HTML

Para testear los siguientes ejemplos deberá utilizar el siguiente código HTML:

```
<!DOCTYPE html>
<html lang="es">va
<head>
  <title>Modelo de caja flexible</title>
  <link rel="stylesheet" href="test.css">
</head>
<body>
<section id="contenedor">
  <div id="caja1">Caja 1</div>
  <div id="caja2">Caja 2</div>
  <div id="caja3">Caja 3</div>
  <div id="caja4">Caja 4</div>
</section>
</body>
</html>
```

Código 2-41

Documento básico HTML para probar el modelo de caja flexible.

Hágalo usted mismo

Cree otro archivo de texto vacío con un nombre y la extensión .html. Copie el texto HTML del Código 2-41 en el archivo. Utilizaremos este modelo para experimentar con las propiedades del Modelo de caja flexible. Las reglas CSS serán incluidas desde un archivo externo llamado test.css. Cree este archivo y añada las normas presentadas más adelante. Compruebe los resultados de la aplicación de todas las reglas abriendo el archivo HTML en su navegador.

Importante

Tenga en cuenta que, por ahora, estas propiedades son experimentales. Deberá declarar cada una añadiendo los prefijos -moz- o -webkit-según el navegador que use. Por ejemplo, display: flex debe escribirse así: display: -webkit-flex para el motor WebKit. No hemos incluido los prefijos en esta parte del capítulo para que el código de origen resulte más fácil de entender.

2.5.3 Display

Un contenedor flexible se define por la propiedad display y se puede ser descrito como un elemento de bloque con el valor flex o como un elemento en línea con el valor inline-flex. Vamos a darle a nuestro contenedor el valor flex.

```
#contenedor {
    display: flex;
}
```

Código 2-42

Declarar el elemento contenedor como un contenedor flexible.

2.5.4 Ejes

Un contenedor flex utiliza ejes para describir la orientación de su contenido. La especificación declara dos ejes que son independientes de la orientación: el eje principal y el eje transversal. En eje principal es distribuido el contenido y por lo general equivale a la orientación horizontal, mientras el eje transversal es perpendicular al eje principal y por lo general equivalente a la orientación. Si la orientación es modificada, los ejes se desplazan junto con el contenido:

Las propiedades de este modelo organizan los elementos desde los extremos de cada eje: inicio del eje principal (main-start), final del eje principal (main-end), inicio del eje transversal (cross-start) y final del eje transversal (cross-end). En general es igual a usar referencias como “de izquierda a derecha” y “de arriba hacia abajo” para describir las direcciones horizontal y vertical, con la diferencia de que la relación se invierte al modificar la orientación. Al mencionar uno de los extremos, por ejemplo main-start, en la descripción de una propiedad, hay que recordar que podría referirse al extremo izquierdo o superior de acuerdo con la orientación actual del contenedor (por ejemplo, en el diagrama de la izquierda, en la Figura 2-4, main-start identifica la parte izquierda del contenedor).

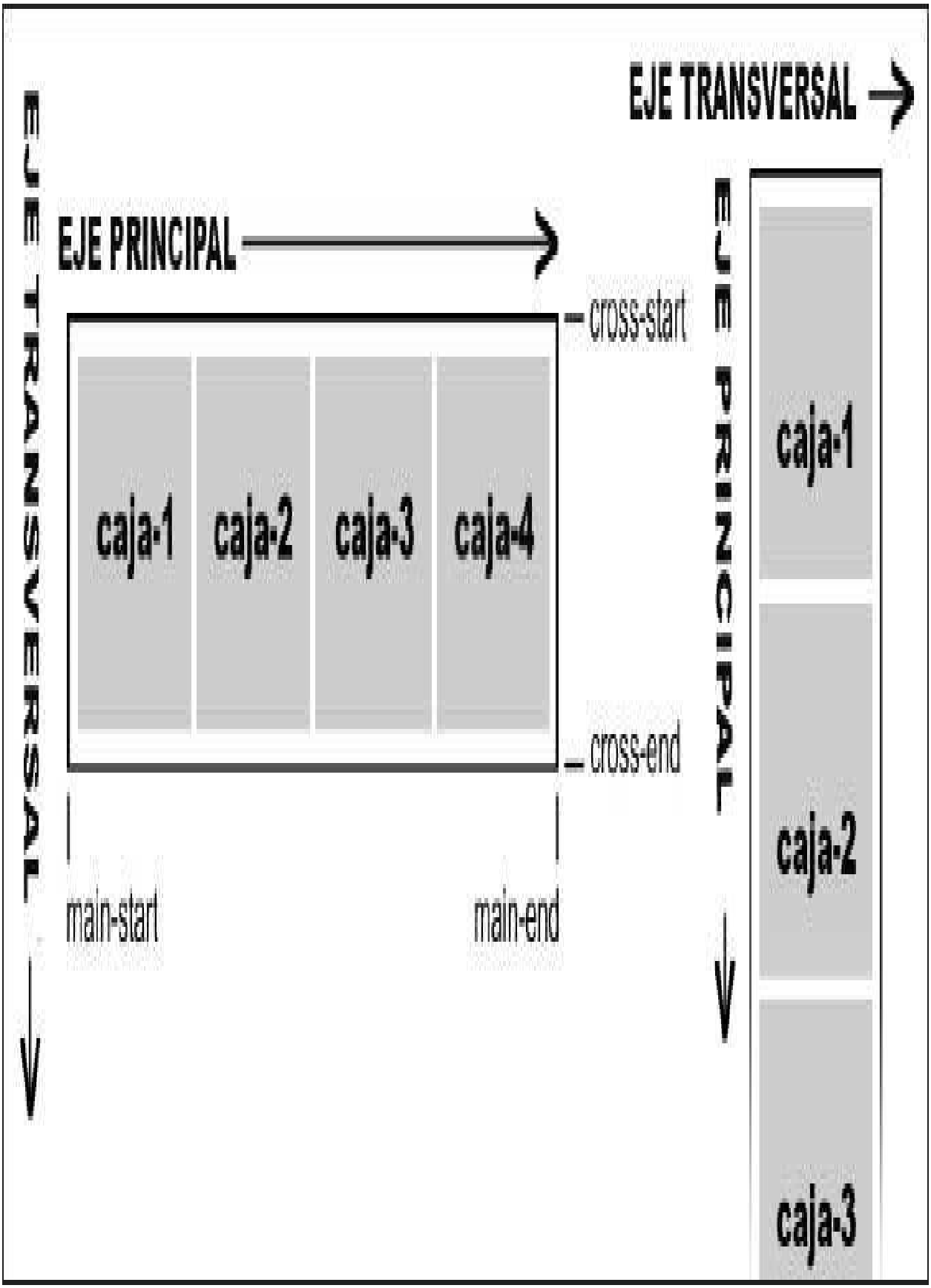

EJE TRANSVERSAL
EJE PRINCIPAL
cross-start
caja-1
caja-2
caja-3
caja-4
cross-end
main-start
main-end
EJE PRINCIPAL
EJE TRANSVERSAL
caja-1
caja-2
caja-3

Figura 2-4

Ejes de contenedores flexibles.

2.5.5 Propiedad Flex

Para que un elemento dentro de un contenedor flexible sea también flexible tiene que ser declarado como tal mediante la propiedad flex. Esta propiedad, que también puede declarar que una caja no es flexible, ayuda a distribuir el espacio entre las cajas. Puede usar tres posibles parámetros, separados por un espacio: flex-grow, flex-shrink y flex-basis. Para declarar una caja como flexible deberá dar un valor mínimo de 1 al primer parámetro, flex-grow. Este parámetro declara la relación de expansión del elemento o, en otras palabras, qué tanto crecerá el elemento dependiendo de la configuración de los elementos hermanos. El segundo parámetro, flex-shrink, declara la relación de reducción o qué tanto podrá ser reducido el elemento según la configuración de sus hermanos. Por último, el tercer parámetro, flex-basis, indica un tamaño inicial para el elemento que será considerado al distribuir el espacio libre entre todos los elementos que se encuentran dentro del mismo contenedor.

Las cajas flexibles, entonces, se amplían o se reducen para rellenar el espacio disponible dentro de su elemento padre. La distribución del espacio dependerá de las propiedades del resto de las cajas. Si todos los elementos del contenedor han sido declarados como flexibles, el tamaño de cada uno de ellos dependerá del tamaño de su elemento padre y del valor de la propiedad flex. Veamos un ejemplo:

```
#contenedor {
   display: flex;
   width: 600px;
}
#caja1{
   flex: 1;
}
#caja2{
   flex: 1;
}
#caja3{
   flex: 1;
}
#caja4{
   flex: 1;
}
```

Código 2-43

Declarar las cajas flexibles con flex.

CAJA CONTENEDORA 600px

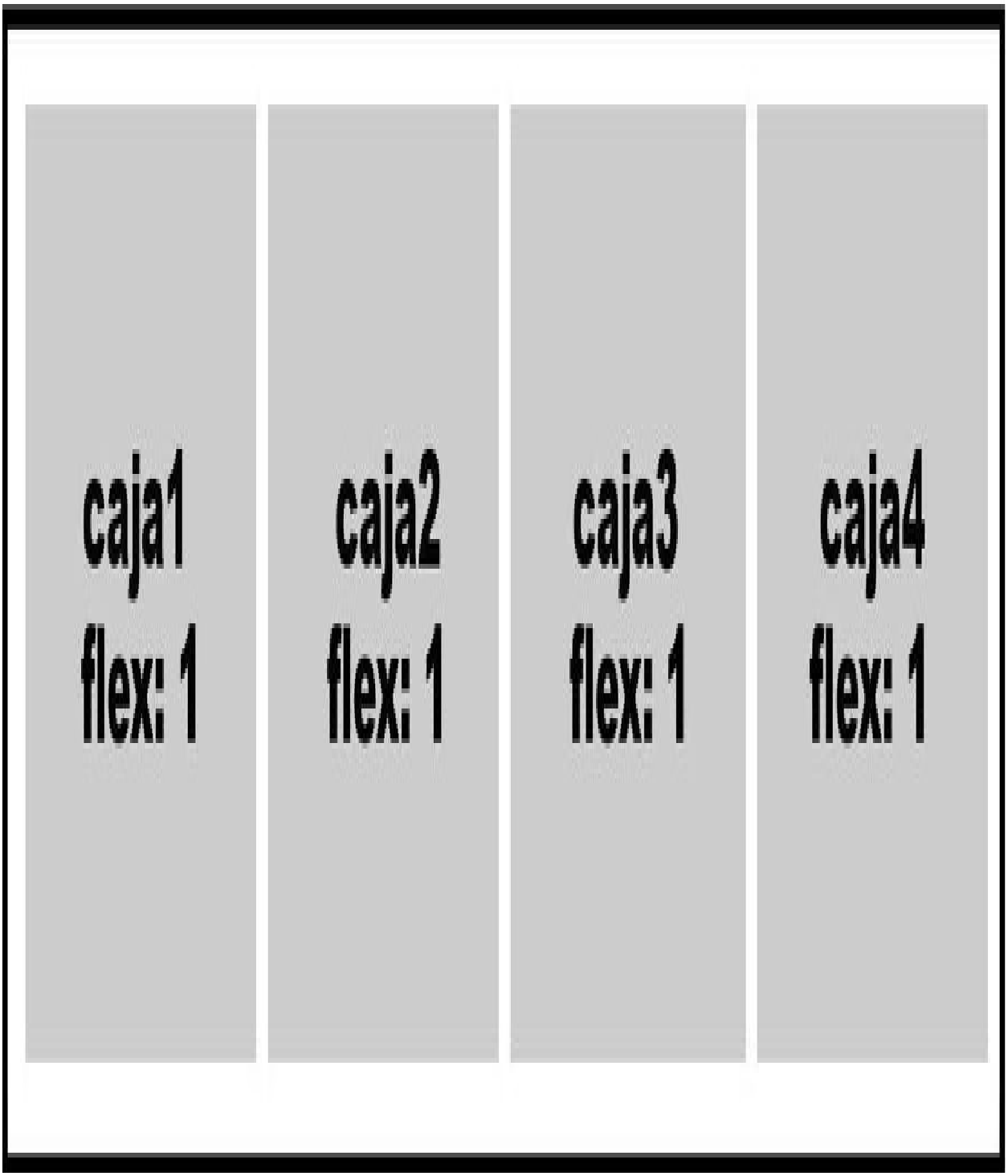


Figura 2-5

El espacio se distribuye de forma equitativa.

En el ejemplo del Código 2-43 declaramos solo el valor flex-grow para la propiedad flex para determinar cómo se ampliarán las cajas. El tamaño de cada caja se calcula multiplicando el valor del tamaño de la matriz por el valor de su propiedad flex y luego dividiendo el resultado entre la suma de los valores flex-grow de todos los elementos hijos. Por ejemplo, la fórmula para el elemento caja1 es de 600 × 1/4 = 150. El valor 600 es el tamaño del contenedor, el 1 es el valor de la propiedad flex para caja1 y el 4 es la suma de los valores de la propiedad flex de cada elemento hijo. Como todas las cajas de nuestro ejemplo tienen un valor igual a 1 para la propiedad flex, el tamaño de todas ellas es 150 px.

Hágalo usted mismo

Copie las reglas CSS del Código 2-43 en el archivo test.css y abra el documento HTML que contiene el Código 2-41 en su navegador. Recuerde que debe añadir los prefijos correspondientes a estas nuevas propiedades antes de la prueba (por ejemplo, -webkit-flex).

El potencial de esta propiedad se hace evidente al proporcionar valores diferentes y realizar combinaciones:

```
#contenedor {
   display: flex;
   width: 600px;
}
#caja1{
   flex: 2;
}
#caja2{
   flex: 1;
}
#caja3{
   flex: 1;
}
#caja4{
   flex: 1;
}
```

Código 2-44

Creación de una distribución desigual.

En el Código 2-44 cambiamos el valor de la propiedad flex para caja1 a 2. Ahora la fórmula para calcular el tamaño de esta caja es: 600 × 2/5 = 240. Como no hemos cambiado el valor del contenedor, el primer valor de la fórmula no varía pero el segundo valor es ahora 2 (el nuevo valor de la propiedad flex para esta caja). Y ahora la suma de los valores de todas las propiedades de los elementos hijos es 5 (2 para la caja 1 y 1 para cada una de las otras tres cajas). La fórmula para calcular el tamaño del resto de los elementos del contenedor es por tanto: 600 × 1/5 = 120.

Al comparar los resultados podemos comprender cómo se distribuye el espacio. El espacio disponible se divide en segmentos de acuerdo a la suma de los valores de la propiedad flex de cada elemento hijo (de un total de 5 en nuestro ejemplo). A continuación, los segmentos se distribuyen entre las cajas. El elemento de caja1 obtiene dos segmentos y el resto de los elementos hijos reciben solo un segmento cada uno, porque el valor de sus propiedades flex es 1.

El efecto de esta propiedad se representa en la Figura 2-6. La ventaja es que cuando se agrega un nuevo elemento al contenedor no es necesario calcular el tamaño que le será asignado porque éste será calculado automáticamente.

CAJA CONTENEDORA 600px

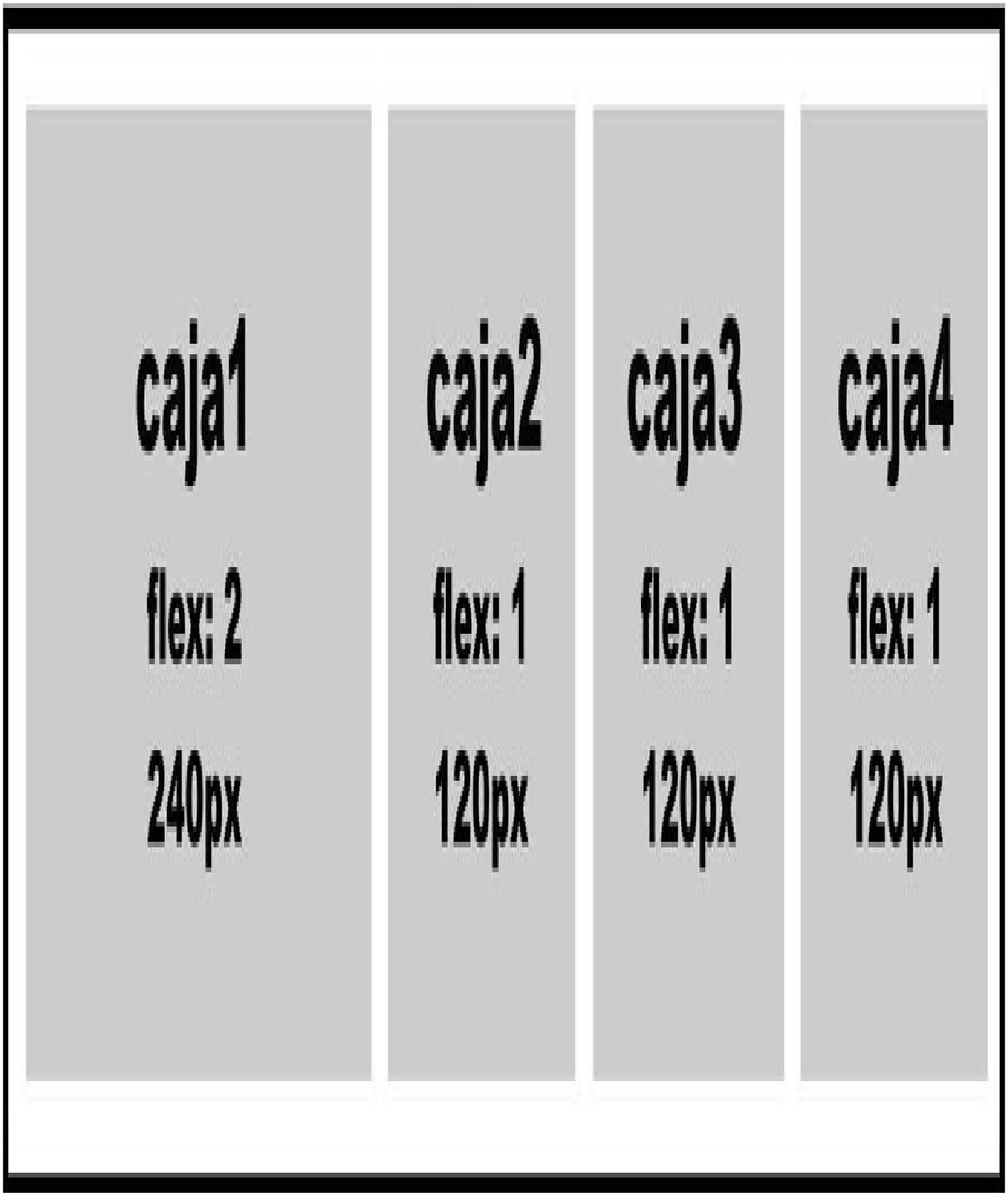


Figura 2-6

El espacio se distribuye de acuerdo con el valor de flex.

Esto es interesante, pero hay más escenarios: imagine que uno de los elementos no es flexible y tiene un tamaño explícito: los otros elementos anidados compartirán el resto del espacio disponible.

```
#contenedor {
   display: flex;
   width: 600px;
}
#caja1{
   width: 300px;
}
#caja2{
   flex: 1;
}
#caja3{
   flex: 1;
}
#caja4{
   flex: 1;
}
```

Código 2-45

Combinación de elementos flexibles y no inflexibles.

La primera caja del Código 2-45 tiene un tamaño de 300 px, por lo que el espacio disponible para distribuir entre el resto de los elementos hijos es de 300 px (es decir: 600 − 300 = 300). El navegador calculará el tamaño de cada caja flexible con la misma fórmula que usamos antes: 300 × 1/3 = 100.

CAJA CONTENEDORA 600px

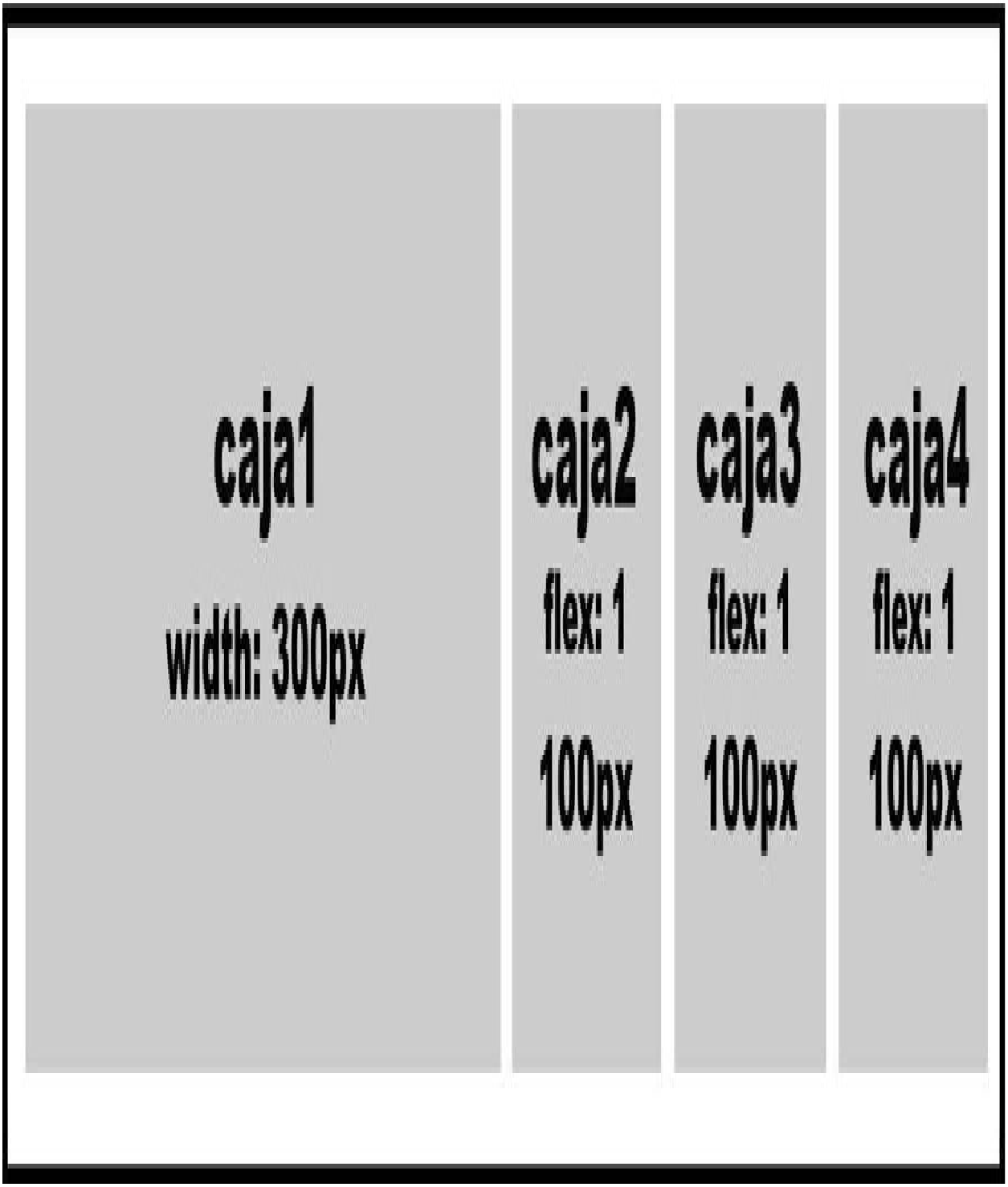


Figura 2-7

Solo se distribuye el espacio libre.

Del mismo modo puede trabajar con varias cajas de tamaño explícito. El principio es el mismo: el espacio libre se distribuirá entre las cajas flexibles.

También es posible que tenga que declarar un tamaño flexible para el elemento y en ese caso será necesario usar el resto de los parámetros disponibles para la propiedad flex (flex-shrink y flex-basis). Observe este ejemplo:

```
#contenedor {
   display: -webkit-flex;
}
#caja1{
   -webkit-flex: 1 1 200px;
}
#caja2{
   -webkit-flex: 1 5 100px;
}
#caja3{
   -webkit-flex: 1 5 100px;
}
#caja4{
   -webkit-flex: 1 5 100px;
}
```

Código 2-46

Controlar cómo reducir los elementos.

En este ejemplo hemos proporcionado tres parámetros para la propiedad flex de cada caja. Todas las cajas tienen el valor 1 para el primer parámetro (flex-grow) y, por tanto, la misma relación de expansión. La diferencia está en los valores de flex-shrink y flex-basis.

Importante

Un valor 0 para las propiedades flex-grow o flex-shrink no permitirá que el elemento sea ampliado o reducido, respectivamente. Para que haya flexibilidad, los valores de estos parámetros deben ser mayores o iguales a 1.

El parámetro flex-shrink funciona de forma similar a flex-grow pero determina la proporción en la que las cajas se reducirán para encajar en el espacio disponible. En nuestro ejemplo, el valor de este parámetro es 1 para la caja 1 pero para el resto de las cajas es igual a 5, lo que asigna un espacio mayor a la caja 1.

El parámetro flex-basis, como hemos mencionado antes, establece un valor inicial para el elemento. Al calcular los valores de reducción o expansión de elementos flexibles, se considera primero el valor del parámetro flex-basis.

Cuando este valor es igual a 0 o no es declarado, el valor tomado en cuenta es el tamaño del contenido de los elementos.

También puede utilizar la palabra clave auto para que el navegador utilice el valor de la propiedad width como referencia, como veremos en el ejemplo que presentaremos en el ejemplo del Código 2-46. En este caso, el parámetro flex-basis tiene el valor auto, así que se establece como tamaño inicial el valor de width.

Hágalo usted mismo

Copie las reglas CSS del Código 2-46 en el archivo test.css para comprobar los efectos de las diferentes combinaciones de los tres parámetros de la propiedad flex. Como no hemos declarado el tamaño del elemento contenedor, cuando la ventana del navegador se expanda el elemento padre se ampliará también y todos los elementos crecerán en la misma proporción. Por el contrario, cuando el tamaño de la ventana se reduzca, el elemento caja1 se reducirá en otra proporción, ya que el valor especificado para flex-shrink es 1 en lugar de 5. Le recomendamos de nuevo añadir otras propiedades a los elementos, como color de fondo o un borde, para obtener una mejor apariencia y poder identificar cada caja en pantalla.

```
#contenedor {
  display: flex;
  width: 600px;
}
#caja1{
  width: 200px;
  flex: 1 1 auto;
}
#caja2{
  width: 100px;
  flex: 1 1 auto;
}
#caja3{
  width: 100px;
  flex: 1 1 auto;
}
#caja4{
  width: 100px;
  flex: 1 1 auto;
}
```

Código 2-47

Definición de cajas flexibles con el tamaño deseado.

En el Código 2-47, cada caja tiene el valor de la propiedad width como primera opción, pero después de que todas las cajas son colocadas, sobra un espacio de 100 px. Este espacio adicional se divide entre las cajas flexibles. Para calcular el espacio asignado a cada casilla se utiliza la misma fórmula: 100 × 1/4 = 25. Esto significa que se añadirán 25 px al ancho inicial de cada caja.

CAJA CONTENEDORA 600px

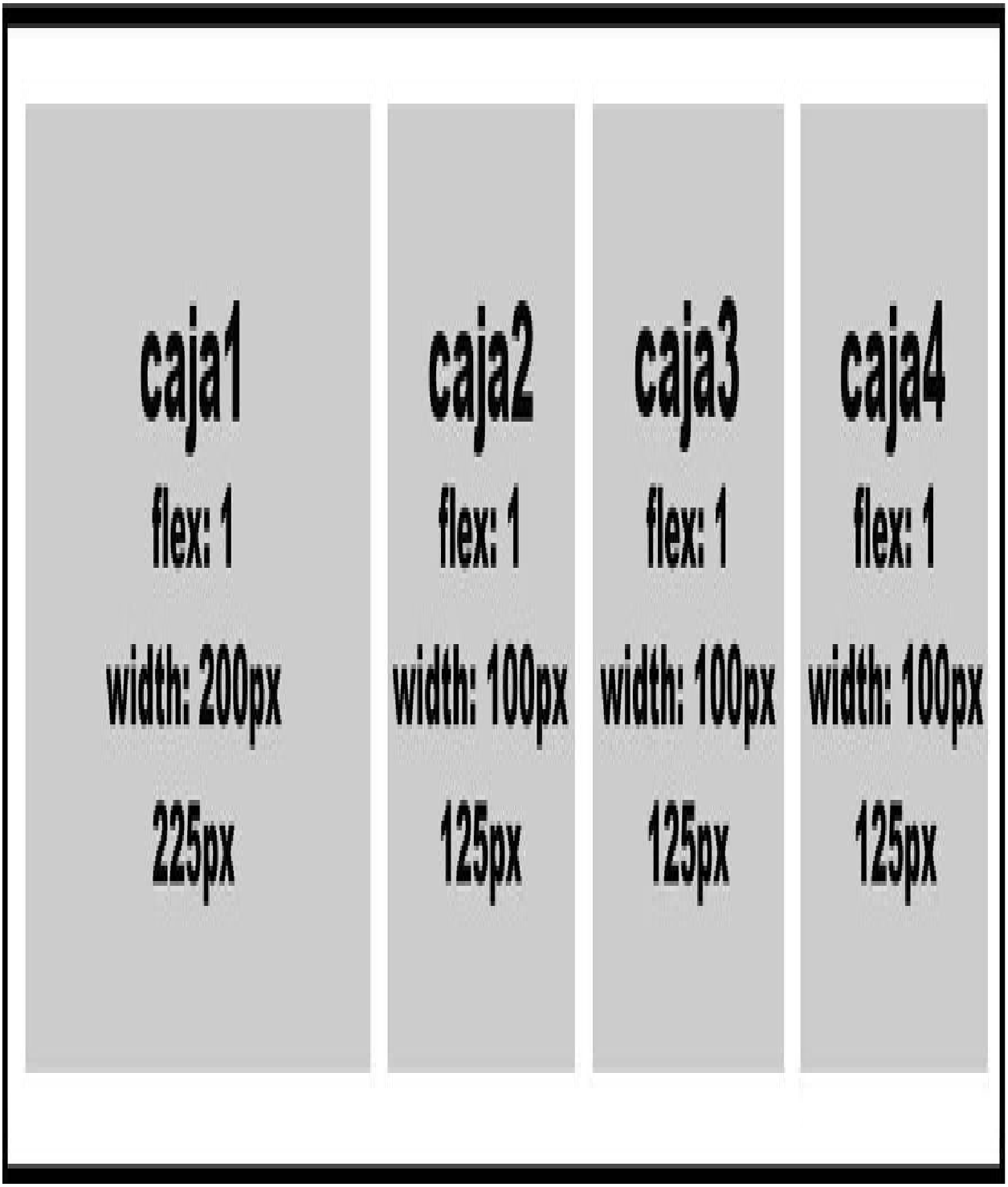


Figura 2-8

Añadir el espacio libre al ancho de cada caja.

2.5.6 flex-direction

Si desea personalizar el orden y la orientación del contenido de un contenedor flexible, tiene que utilizar la propiedad flex-direction, que tiene cuatro valores posibles: row, row-reverse, column y column-reverse, con row como valor predeterminado.

row: Este valor fija la orientación de acuerdo con la orientación del texto (generalmente horizontal) y ordena a los elementos dentro del contenedor desde el inicio hasta el final del eje principal (por lo general de izquierda a derecha).

row-reverse: Este valor establece la misma orientación que row, pero invierte el orden de los elementos desde el final del eje principal hacia el inicio del mismo (generalmente de derecha a la izquierda).

column: Este valor fija la orientación de acuerdo a la forma en la que se distribuyen los bloques de texto (generalmente de forma vertical) y ordena a los hijos desde el inicio del eje principal hacia el final de éste (por lo general de arriba a abajo).

column-reverse: Este valor establece la misma orientación que column pero invierte el orden de los elementos, que por tanto es desde el final hacia el inicio del eje principal (generalmente de abajo hacia arriba).

```
#contenedor {
   display: flex;
   flex-direction: row;
}
#caja1{
   flex: 1;
}
#caja2{
   flex: 1;
}
#caja3{
   flex: 1;
}
#caja4{
   flex: 1;
}
```

Código 2-48

Cambio de la orientación de los hijos.

Conceptos básicos

La propiedad writing-mode establece que la orientación de las líneas de texto sea horizontal o vertical. Por esta razón el resultado de las propiedades del Modelo de caja flexible dependerá de la orientación previamente establecida para el texto. Para obtener más información sobre esta propiedad, por favor visite nuestro sitio web y siga los enlaces de este capítulo.

2.5.7 order

El orden de los elementos hijos también puede ser personalizado. La propiedad order permite declarar un lugar específico para cada caja:

```
#contenedor {
   display: flex;
}
#caja1{
   flex: 1;
   order: 2;
}
#caja2{
   flex: 1;
   order: 4;
}
#caja3{
   flex: 1;
   order: 3;
}
#caja4{
   flex: 1;
   order: 1;
}
```

Código 2-49

Definición de la posición de cada cuadro.

Como se puede ver en el Código 2-49, la propiedad order debe ser asignada a los elementos hijos. Si el valor se duplica, las cajas afectadas seguirán el orden del código fuente.

CAJA CONTENEDORA

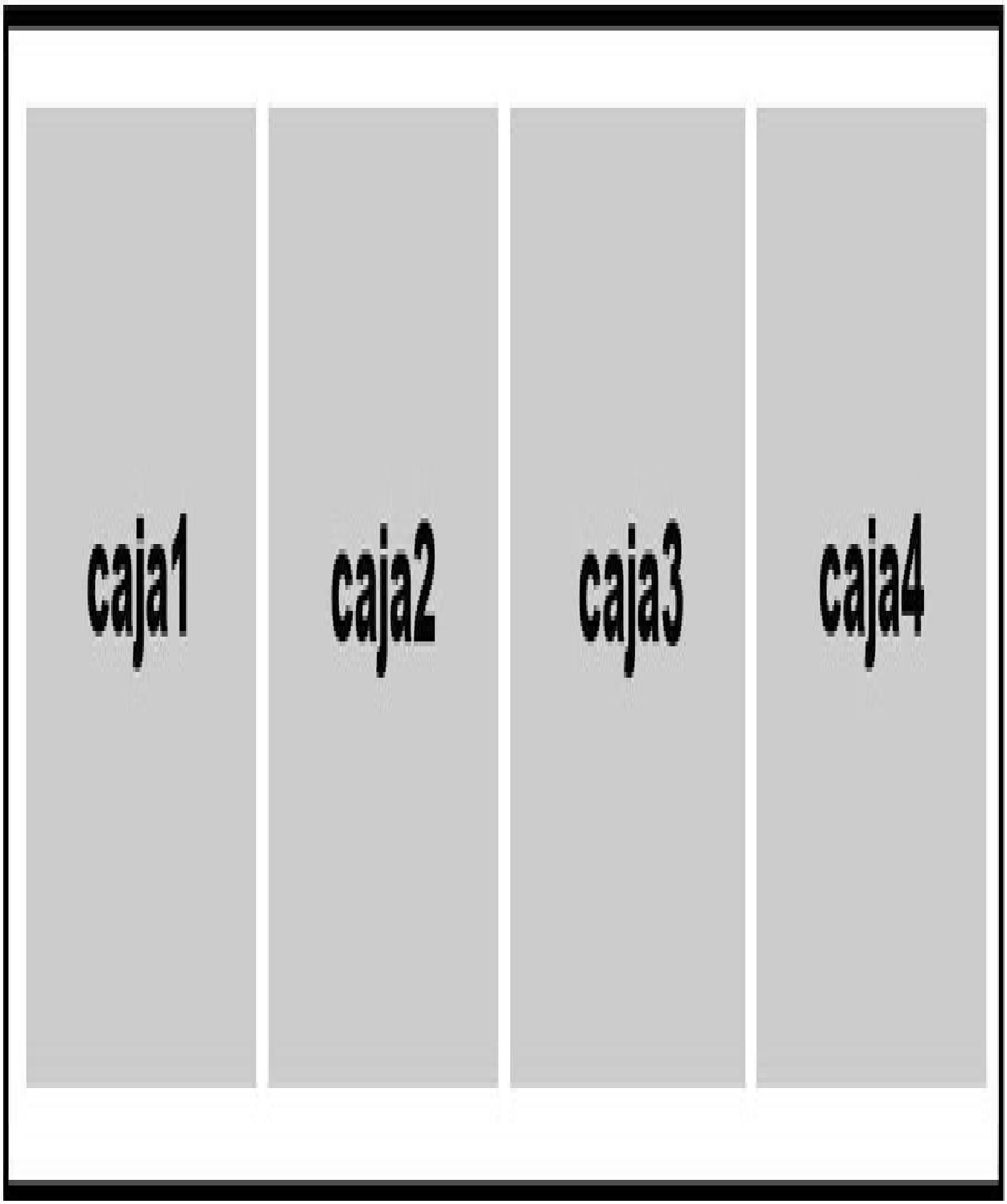


Figura 2-9

Posiciones específicas para cada caja de acuerdo al Código 2-49.

2.5.8 justify-content

Otra de las características más relevantes del Modelo de caja flexible es la capacidad para diseñar el espacio libre. Cuando el tamaño de los elementos hijos no llena todo al contenedor, queda un espacio libre que tiene que ser colocado en algún lugar en el diseño. Veamos un ejemplo:

```
#contenedor {
   display: flex;
   width: 600px;
}
#caja1{
   width: 100px;
}
#caja2{
   width: 100px;
}
#caja3{
   width: 100px;
}
#caja4{
   width: 100px;
}
```

Código 2-50

Distribución del espacio libre en un contenedor flexible.

Hágalo usted mismo

Agregue otras propiedades a los elementos, como fondo de color, altura o borde, para poder ver e identificar a cada uno en pantalla.

En el Código 2-50 el tamaño de nuestro contenedor flexible es 600 px y cada elemento hijo tiene 100 px de ancho. Por tanto, hay 200 px de espacio libre.

Podemos elegir diferentes formas de distribuir el espacio libre. Por defecto, los elementos hijos están ordenados de acuerdo a la Figura 2-10, de inicio a fin del eje principal (por lo general de izquierda a derecha), dejando el espacio libre al final. Sin embargo, la propiedad justify-content ofrece alternativas, pues establece cómo se distribuyen en el eje principal de un contenedor flexible tanto los elementos hijos como el espacio sobrante. Esta propiedad puede tomar cinco valores: flex-start, flex-end, center, space-between and space-around. Si la propiedad no es declarada, se aplica el valor flex-start por defecto, que alinea los elementos hacia el inicio del eje principal y el espacio libre hacia el final del eje principal (de izquierda a derecha o de arriba hacia abajo).

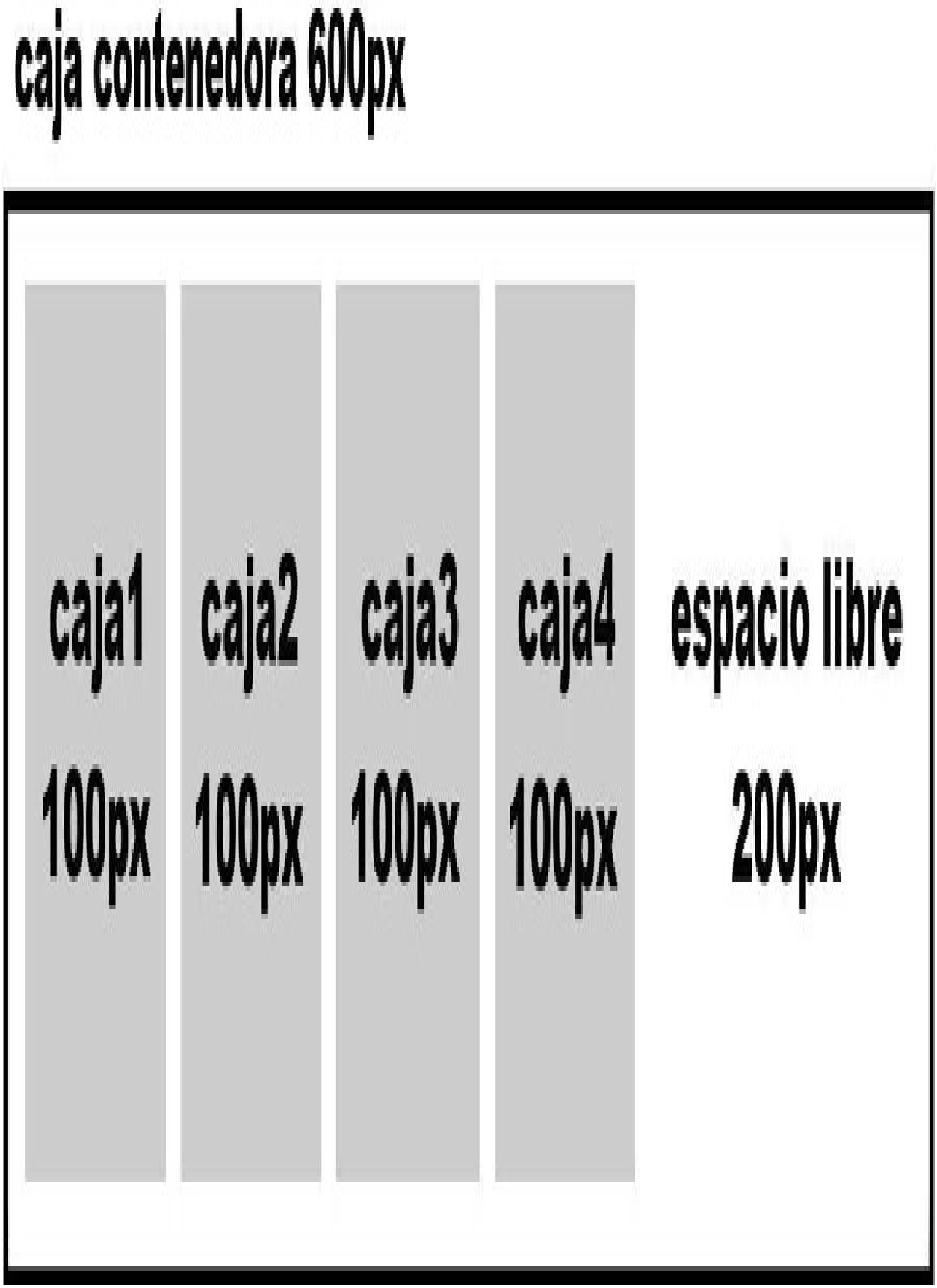

caja contenedora 600px
caja1
100px
caja2
100px
caja3
100px
caja4
100px
espacio libre
200px

Figura 2-10

Las cajas y el espacio libre en el interior de un contenedor flexible.

```
#contenedor {
   display: flex;
   width: 600px;
   justify-content: flex-end;
}
#caja1{
   width: 100px;
}
#caja2{
   width: 100px;
}
#caja3{
   width: 100px;
}
#caja4{
   width: 100px;
}
```

Código 2-51

Distribución del espacio libre con justify-content.

Las siguientes figuras muestran el potencial de esta propiedad y el efecto del modelo de caja flexible.

caja contenedora 600px

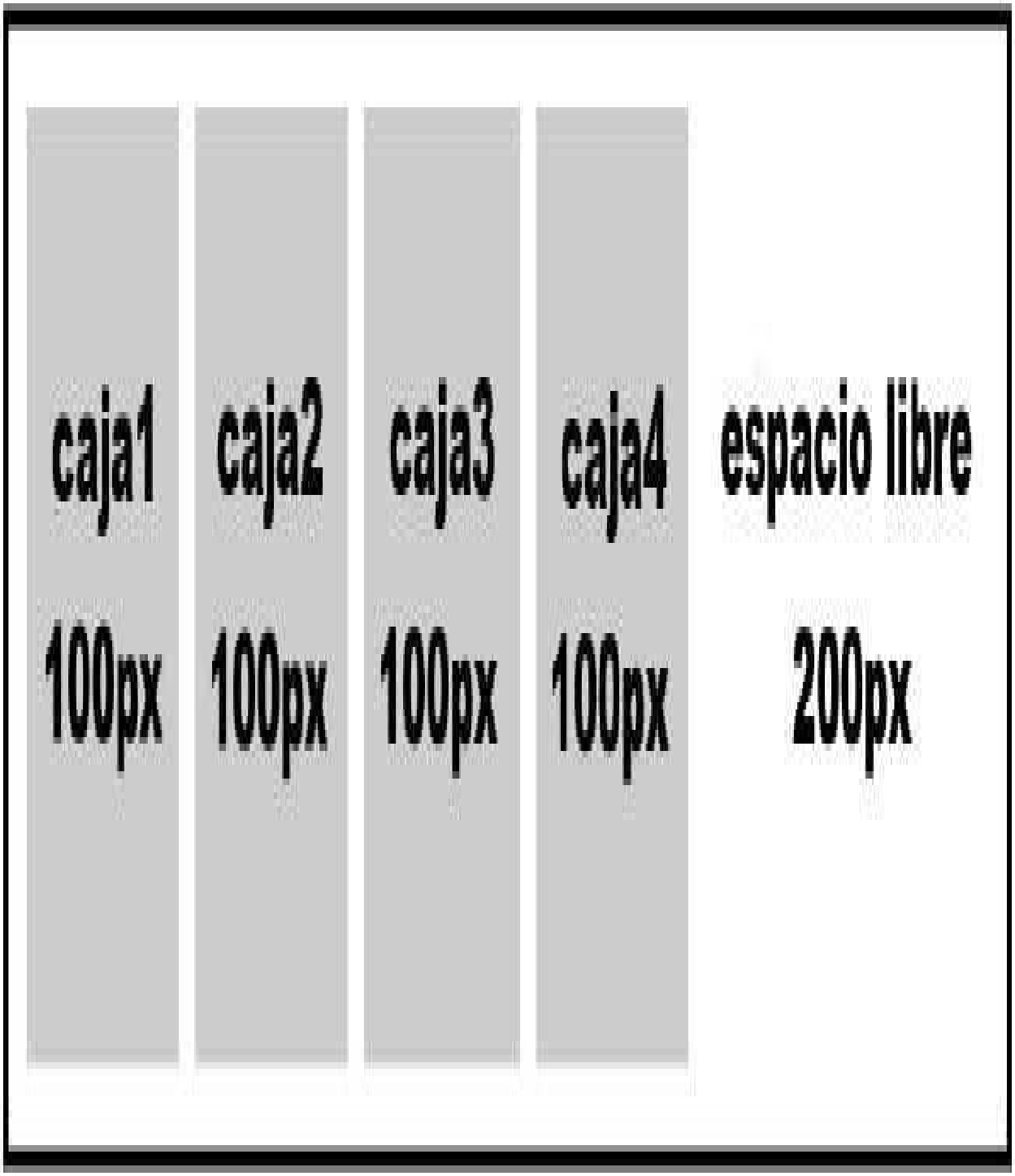


Figura 2-11

Distribución del espacio libre con justify-content: flex-start.

caja contenedora 600px

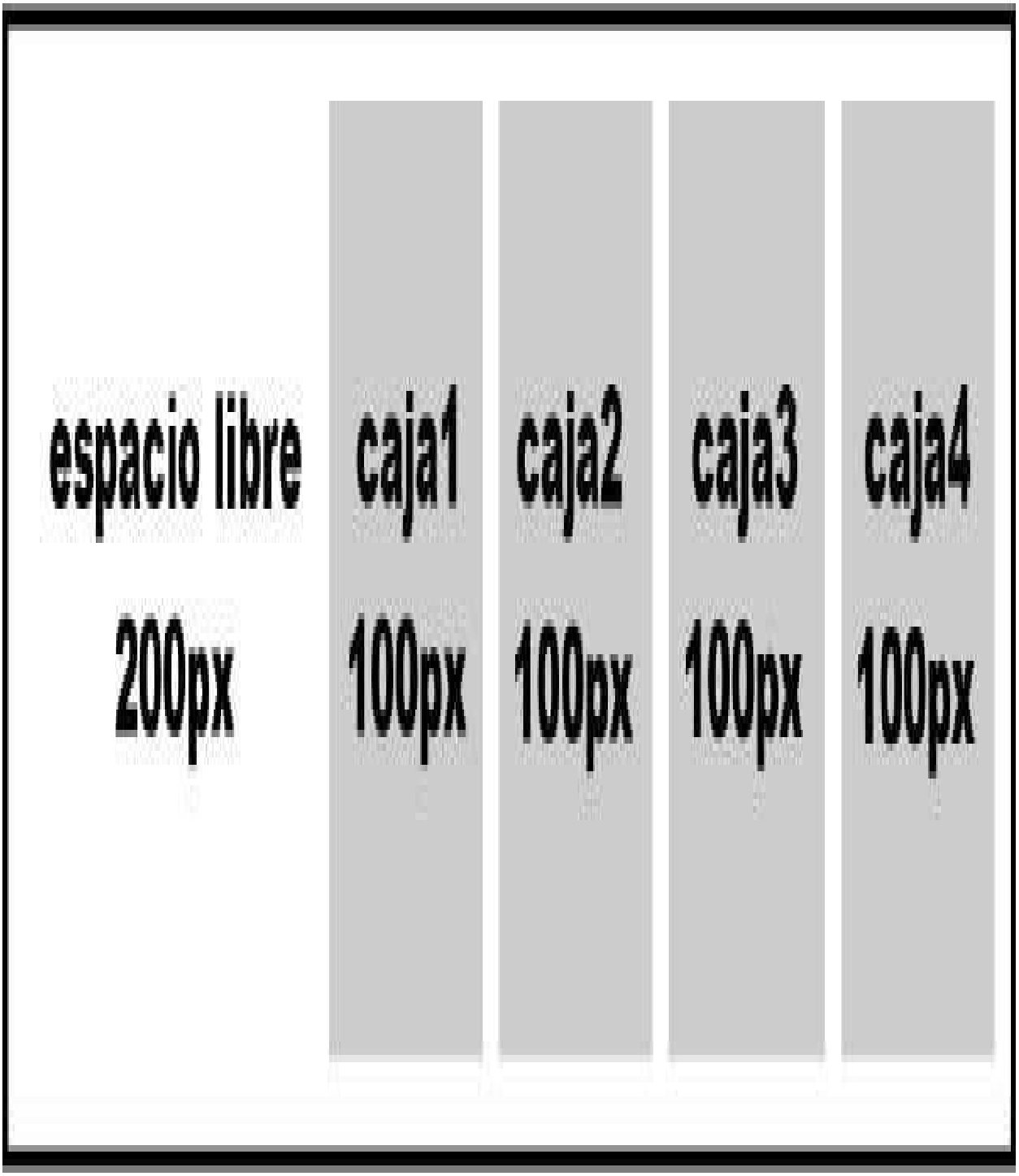


Figura 2-12

Distribución del espacio libre con justify-content: flex-end.

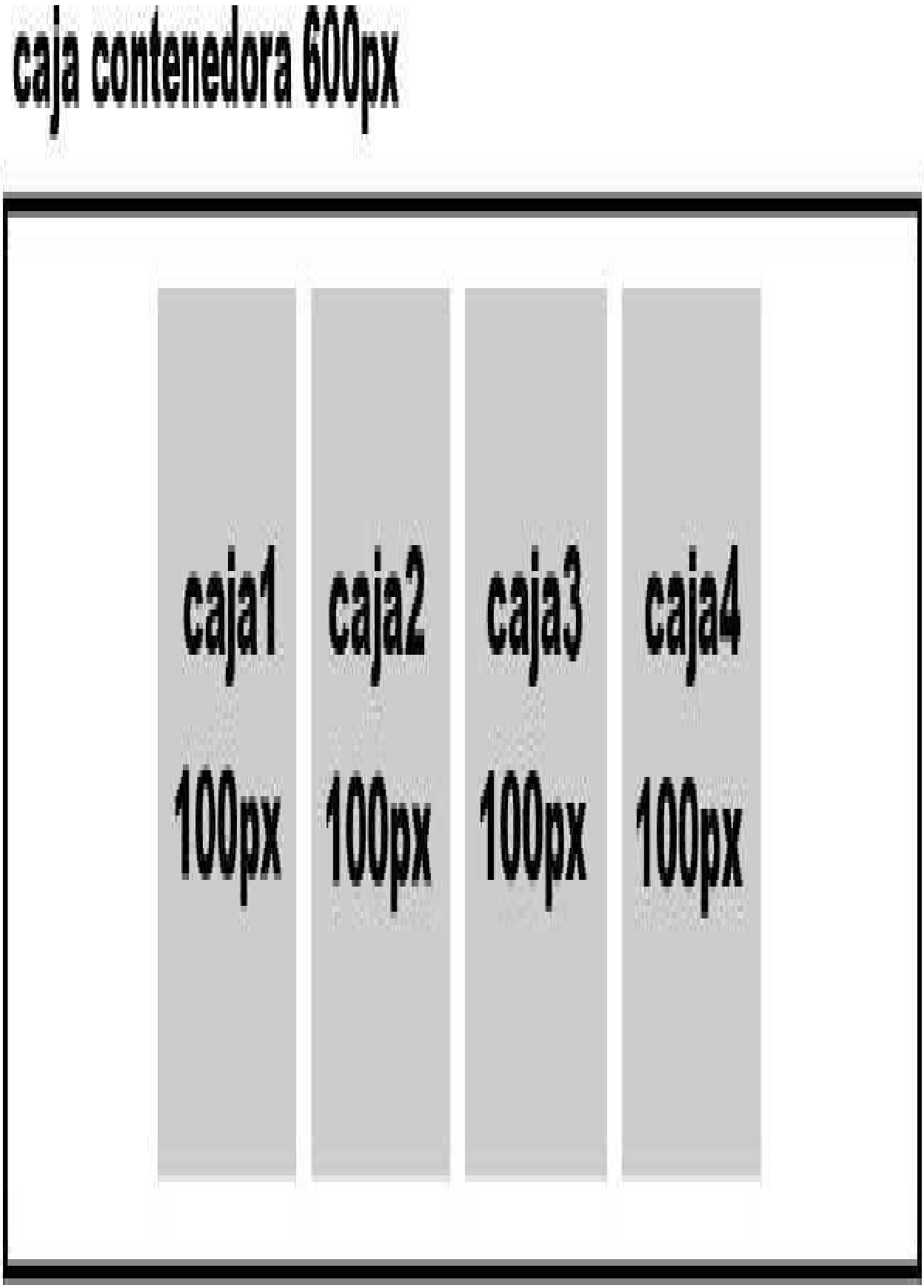

caja contenedora 600px
caja1
100px
caja2
100px
caja3
100px
caja4
100px

Figura 2-13

Distribución del espacio libre con justify-content: center.

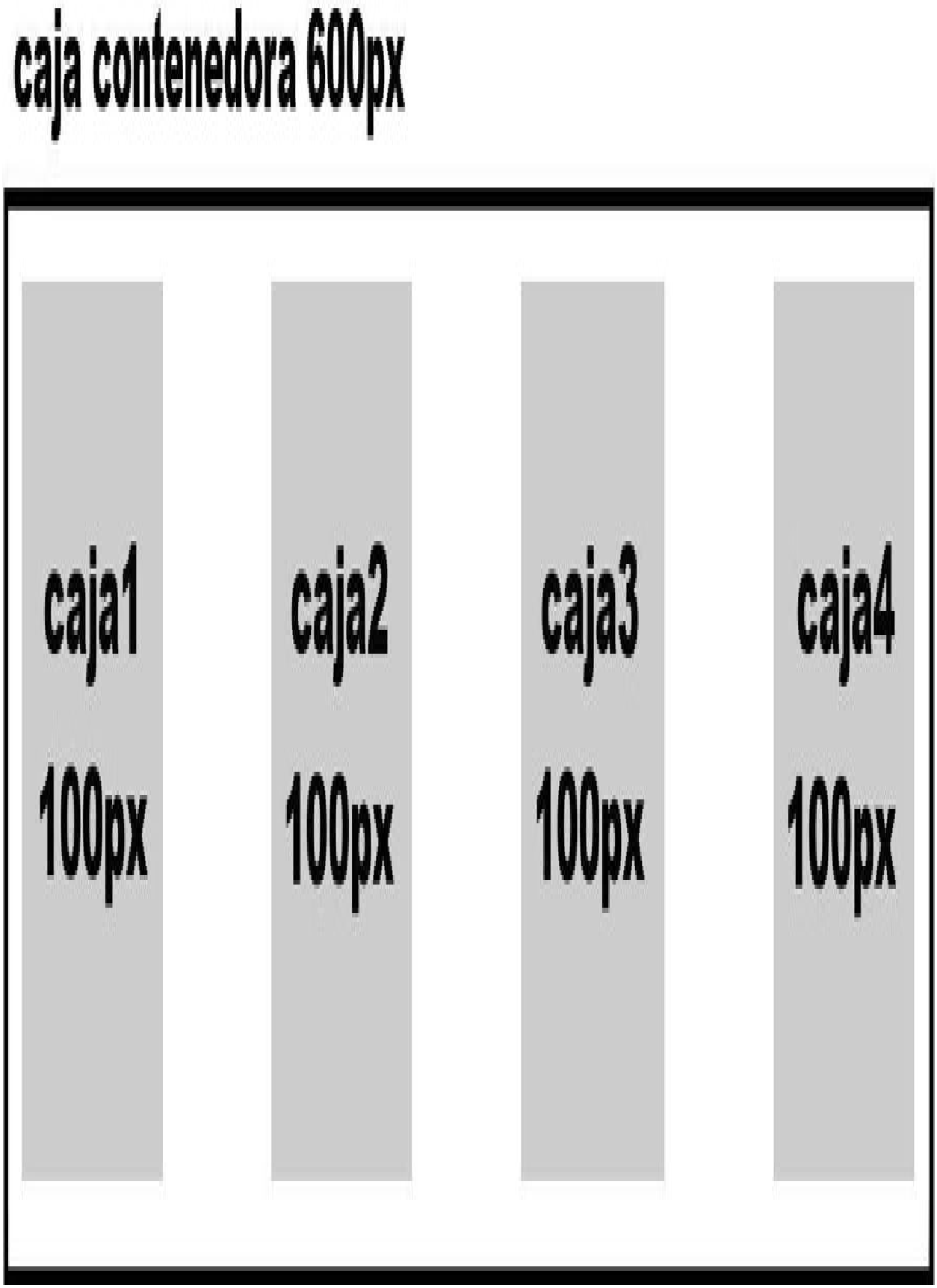

caja contenedora 600px
caja1
100px
caja2
100px
caja3
100px
caja4
100px

Figura 2-14

Distribución del espacio libre con justify-content: space-between.

caja contenedora 600px

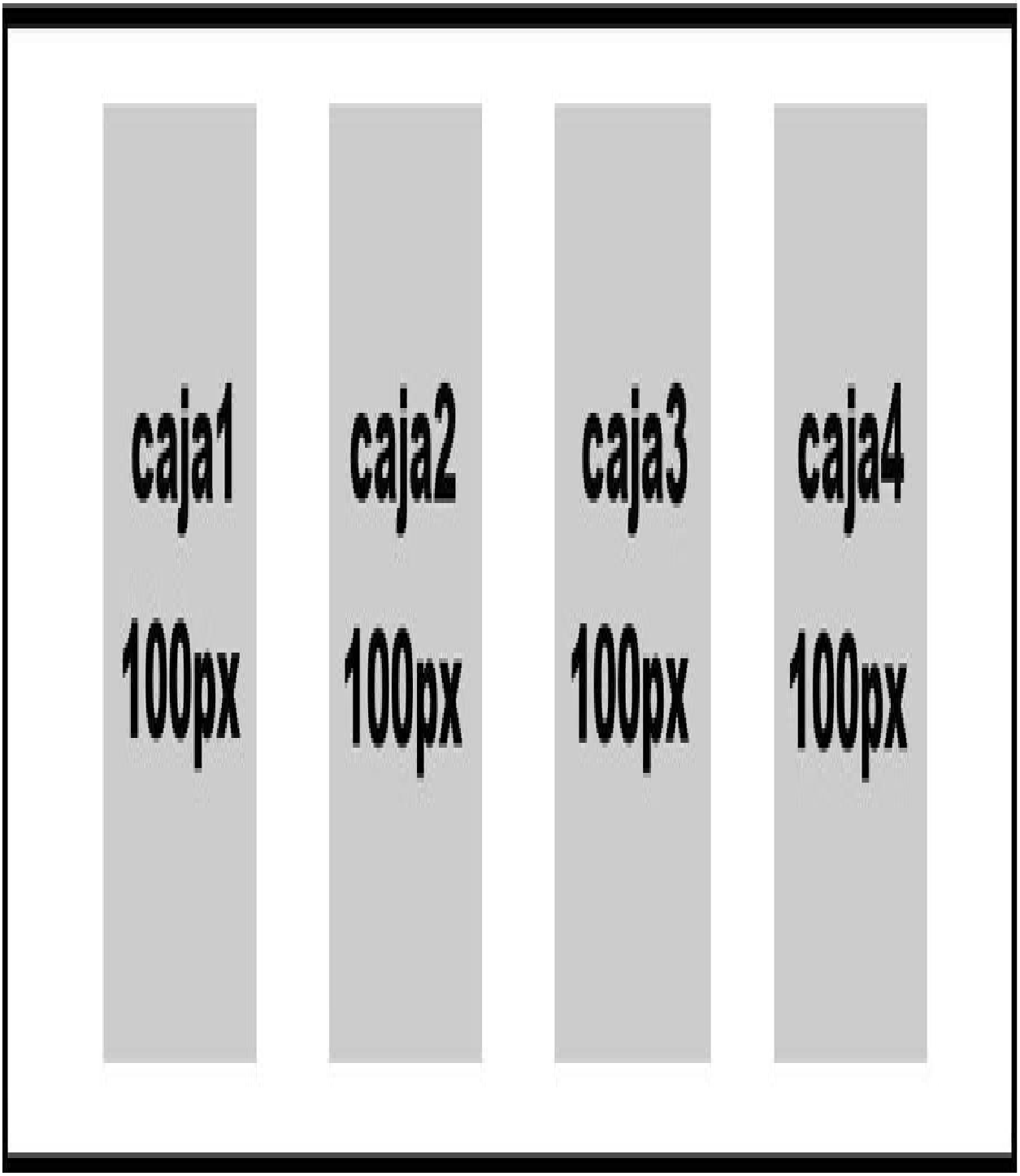


Figura 2-15

Distribución del espacio libre con justify-content: space-around.

2.5.9 align-items

Otra propiedad que nos ayudará a distribuir el espacio es align-items. Esta propiedad funciona como justify-content pero alinea las cajas en el eje transversal, por tanto es adecuada para la alineación vertical, poco habitual debido a que el uso de las tablets ha sido menospreciado.

```
#contenedor {
  display: flex;
  align-items: center;
  width: 600px;
  height: 200px;
}
#caja1{
  flex: 1;
  height: 100px;
}
#caja2{
  flex: 1;
  height: 100px;
}
#caja3{
  flex: 1;
  height: 100px;
}
#caja4{
  flex: 1;
  height: 100px;
}
```

Código 2-52

Distribuir el espacio vertical.

En el Código 2-52 ofrecemos una altura específica para cada caja, incluyendo el contenedor, y queda un espacio libre de 100 px que será establecido de acuerdo con el valor de la propiedad align-items.

caja contenedora 200px de altura

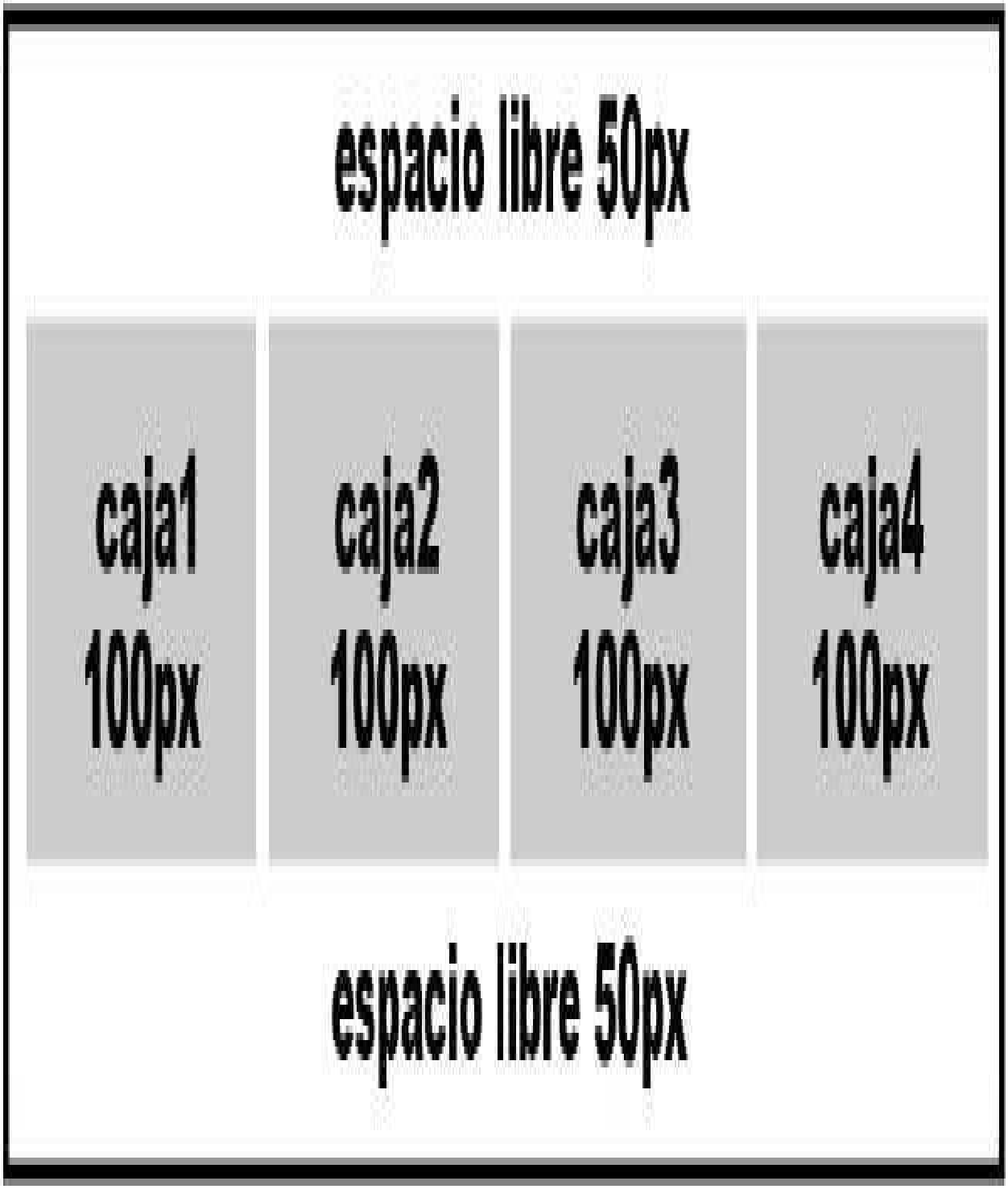


Figura 2-16

La alineación vertical con align-items: center.

La propiedad align-items admite los siguientes valores: flex-start, flex-end, center, baseline y stretch. El último valor estira las cajas de arriba a abajo para ajustar los elementos hijos al espacio disponible y es una característica tan importante que es el valor establecido de forma predeterminada al crear un recipiente flexible. El stretch establece que si la altura de los elementos hijos no es declarada, éstos adoptan automáticamente la altura de su contendor.

caja contenedora 200px de altura

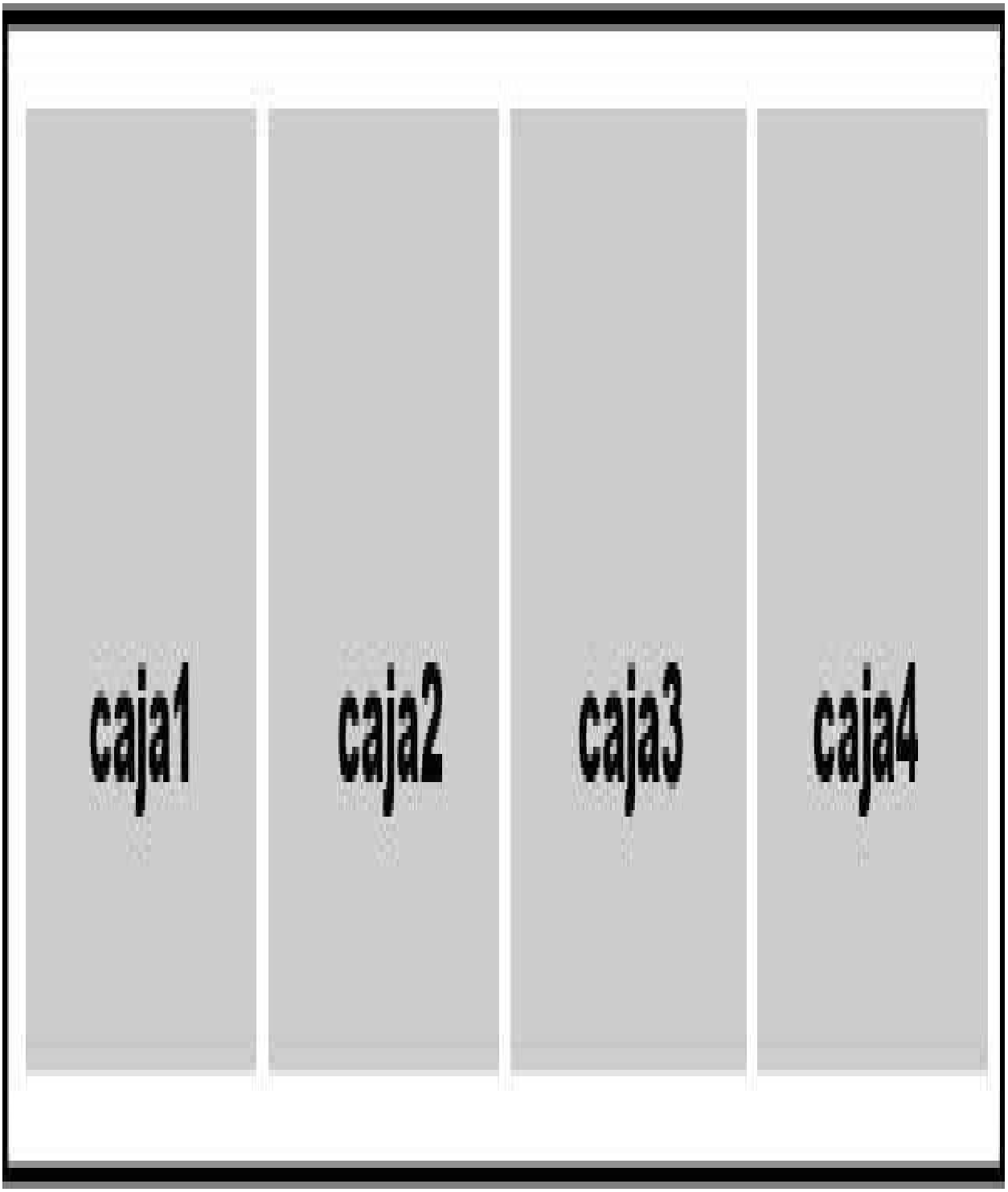


Figura 2-17

Elementos anidados estirados con align-items: stretch.

Esta propiedad es extremadamente útil cuando el diseño tiene columnas con diferentes cantidades de contenido, que de otro modo harían que unos elementos fueran más cortos que los otros. Usando el valor stretch para la propiedad align-items, el tamaño de las columnas más cortas coincidirá con el tamaño de la más larga.

Por otra parte, el valor flex-start alinea las cajas en el inicio de la línea, que como recordará está determinado por la orientación del contenedor (por lo general a la izquierda o arriba):

caja contenedora 200px de altura

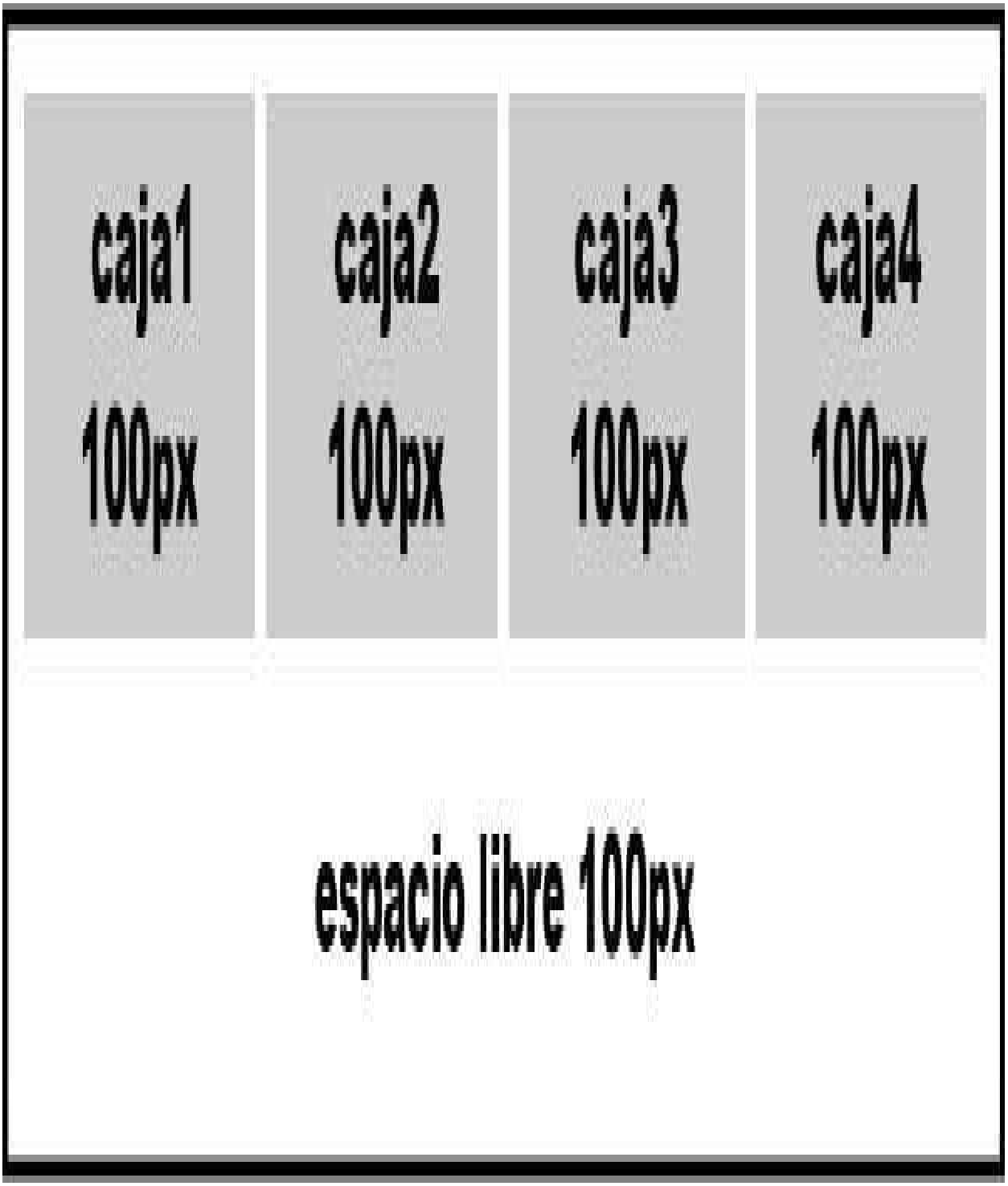


Figura 2-18

Alineación de las cajas con align-items: flex-start.

El valor flex-end alinea las cajas al final del contenedor (normalmente abajo o a la derecha):

caja contenedora 200px de altura

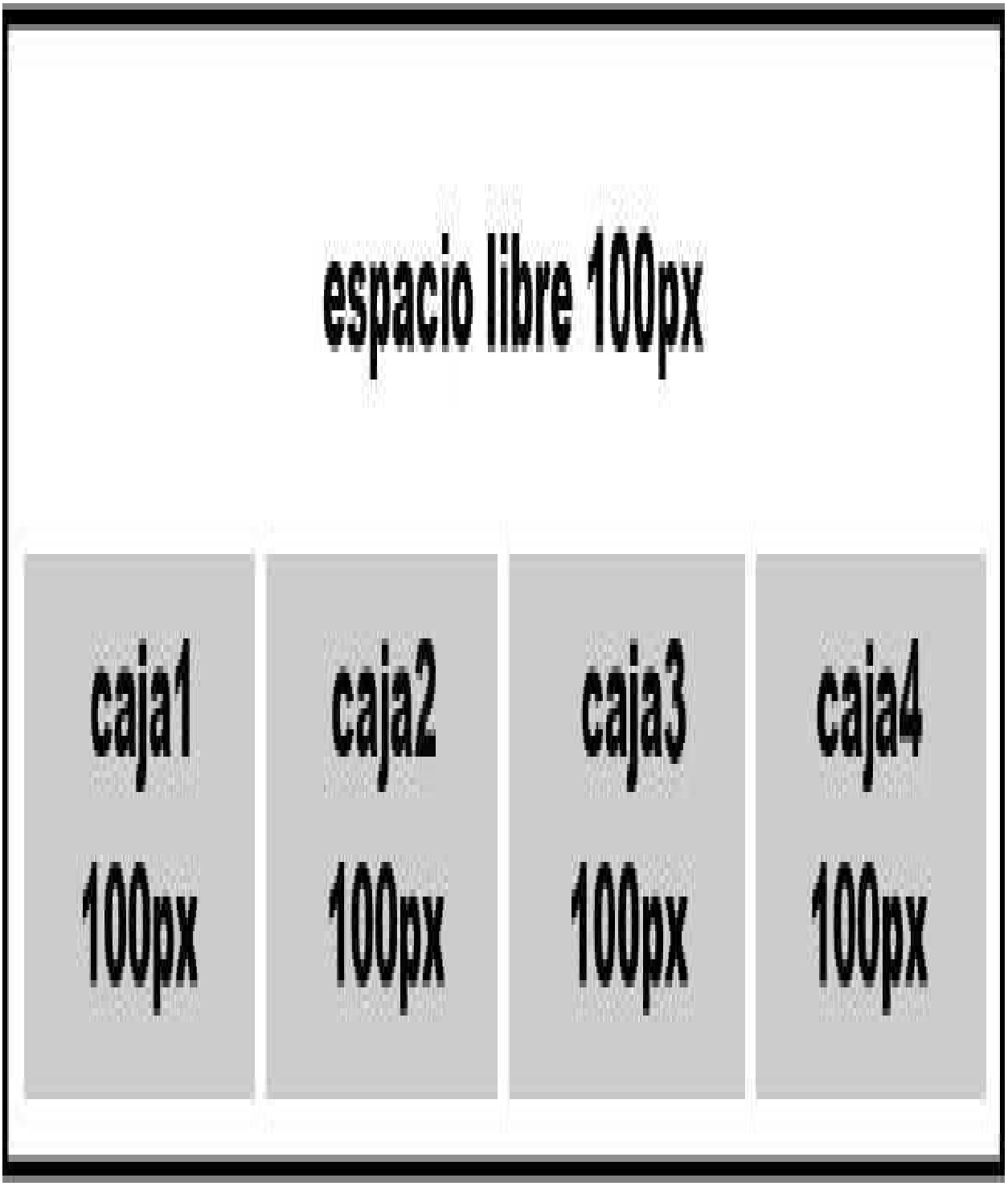


Figura 2-19

Alineación de elementos hijos con align-items: flex-end.

Por último, el valor baseline alinea las cajas desde base de la primera línea de contenido:

caja contenedora 200px de altura

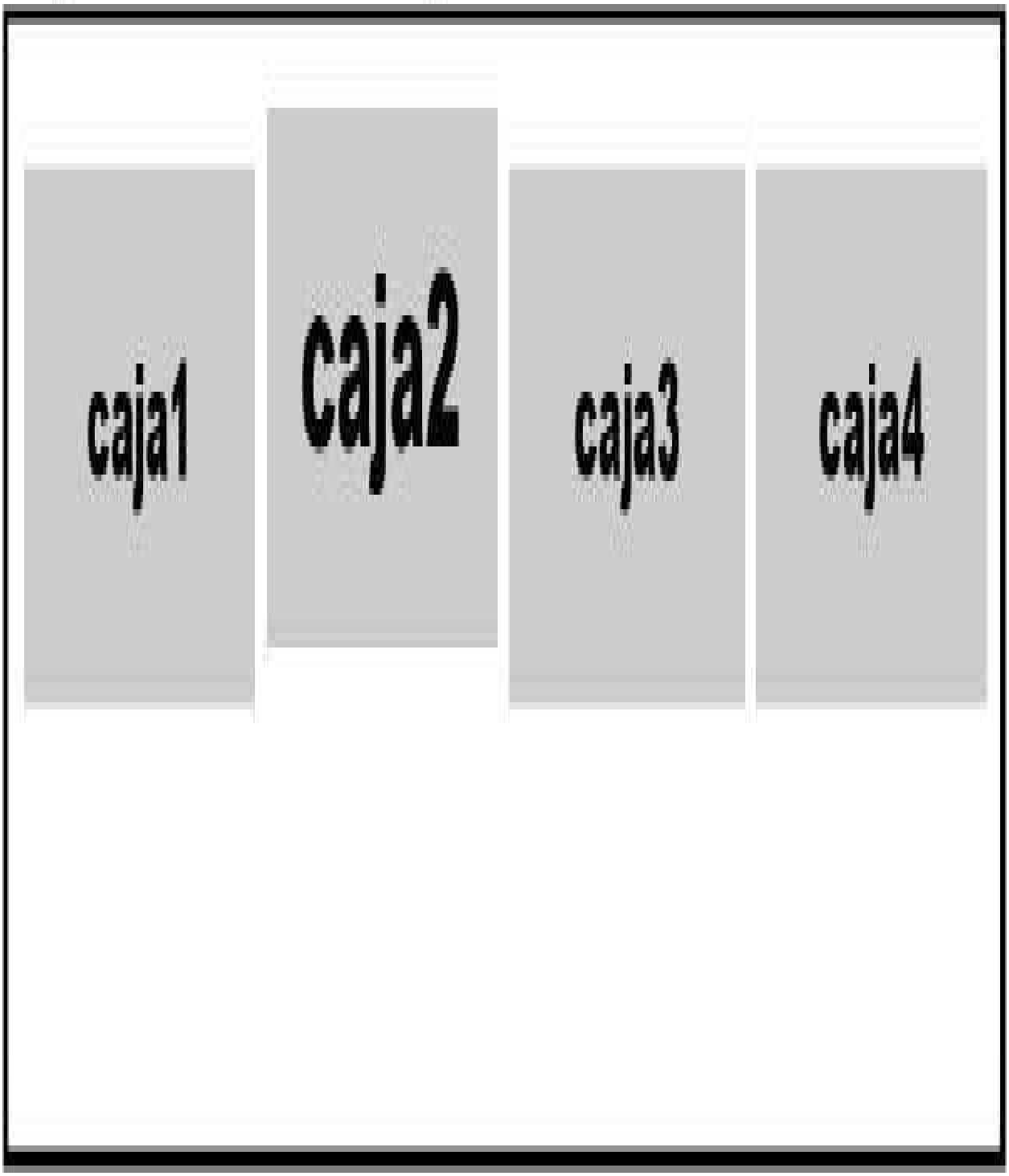


Figura 2-20

Alineación de elementos hijos con align-items: baseline.

Hágalo usted mismo

Utilice las reglas CSS del Código 2-52 y pruebe diferentes valores para la propiedad align-item. Borre la propiedad height para cada elemento e introduzca algún contenido para caja1. Use el valor de stretch para la propiedad align-items (o no declarare esta propiedad) para ver cómo el resto de los cuadros se amplían para que su altura coincida con la altura de caja1. También puede declarar la propiedad font-size: 50px para el elemento caja2 como se muestra en la Figura 2-20, para probar cómo el valor baseline alinea las cajas desde la base de la primera línea de contenido.

2.5.10 align-self

A veces puede resultar útil alinear las cajas independientemente de la alineación establecida por sus elementos padres o contenedores. La propiedad de align-self funciona exactamente igual align-items, pero para elementos hijos.

```
#contenedor {
  display: flex;
  align-items: flex-end;
  width: 600px;
  height: 200px;
}
#caja1{
  flex: 1;
  height: 100px;
}
#caja2{
  flex: 1;
  height: 100px;
  align-self: center;
}
#caja3{
  flex: 1;
  height: 100px;
}
#caja4{
  flex: 1;
  height: 100px;
}
```

Código 2-53

Cambio de alineación para caja2.

Las reglas CSS del Código 2-53 alinean los elementos en la parte inferior del contenedor flexible, pero la propiedad align-self desplaza el elemento de caja2 al centro, tal como se muestra en la Figura 2-21.

caja contenedora 200px de altura

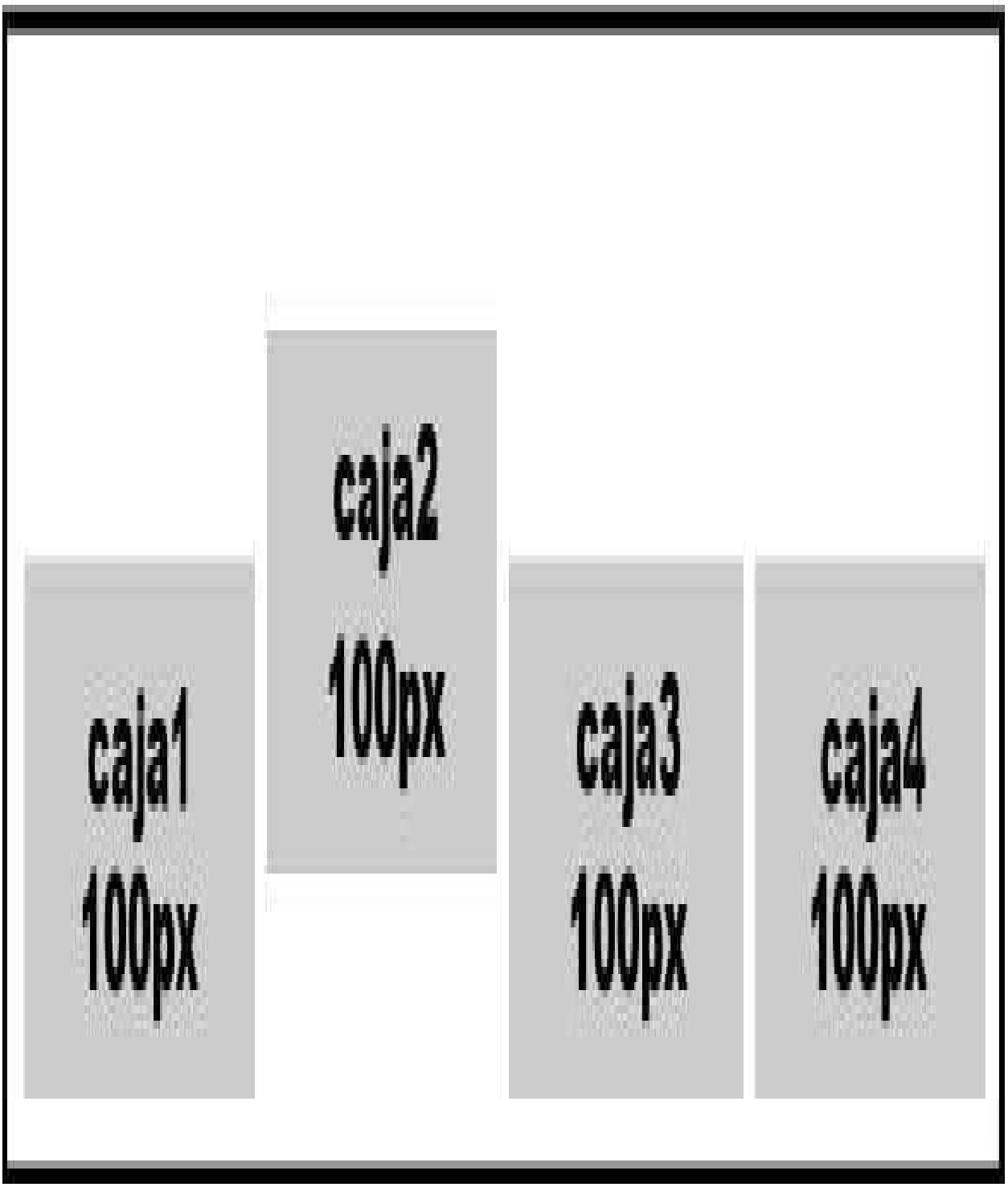


Figura 2-21

Usar align-self para alinear una caja en particular.

2.5.11 flex-wrap

Un contenedor flexible puede tener una o más líneas de cajas. La propiedad flex-wrap declara esta condición mediante tres valores posibles: nowrap, wrap y wrap-reverse. El valor nowrap establece un contenedor flexible de una sola línea, wrap declara un contenedor de varias líneas y ordena a las líneas desde el inicio del eje transversal hasta el final de éste, generalmente de izquierda a derecha o de arriba a abajo. El último valor disponible, llamado wrap-reverse, invierte este orden.

```
#contenedor {
   display: flex;
   justify-content: center;
   flex-wrap: wrap;
   width: 600px;
}
#caja1{
   width: 100px;
}
#caja2{
   width: 100px;
}
#caja3{
   width: 100px;
}
#caja4{
   flex: 1 1 400px;
}
```

Código 2-54

La creación de dos líneas de cajas con la propiedad flex-wrap.

En el Código 2-54 las tres primeras cajas tienen un tamaño de 100 px y caben en una sola línea de 600 px de ancho del contenedor, pero la última caja ha sido declarada flexible con un tamaño inicial de 400 px (flex-basis). Como no hay espacio suficiente en el contenedor para mostrar todas las cajas, hay dos opciones: reducir el tamaño del cuadro flexible para que se adapte al espacio disponible o generar una nueva línea. Como en este caso hemos asignado el valor wrap para la propiedad flex-wrap, el navegador aplica la segunda opción:

caja contenedora

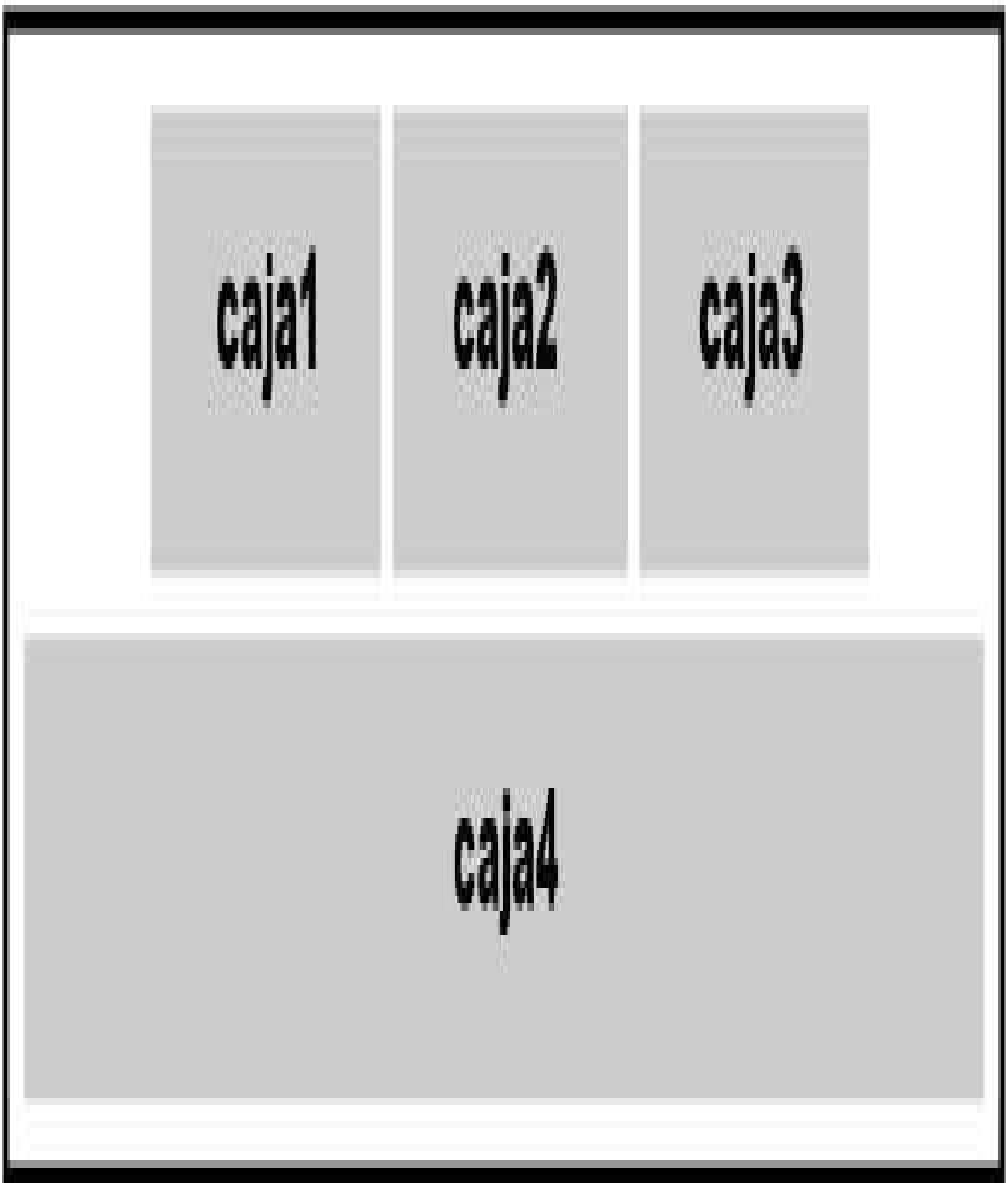


Figura 2-22

Contenedor flexible multilínea.

El elemento caja4 fue declarado lo más flexible por su propiedad flex, por lo que no solo se coloca en una segunda línea, pero también se expandió para ocupar el espacio total disponible en la línea (recuerde que los 400 px declarados por el parámetro flex-basis son solo el valor inicial, no una declaración de tamaño). Además, las cajas en la primera línea, están alineadas con el centro debido al nuevo espacio disponible en la primera línea y el valor de la propiedad justify-content.

El orden de las líneas puede ser invertido con el valor wrap-reverse, como se muestra en la siguiente figura.

caja contenedora

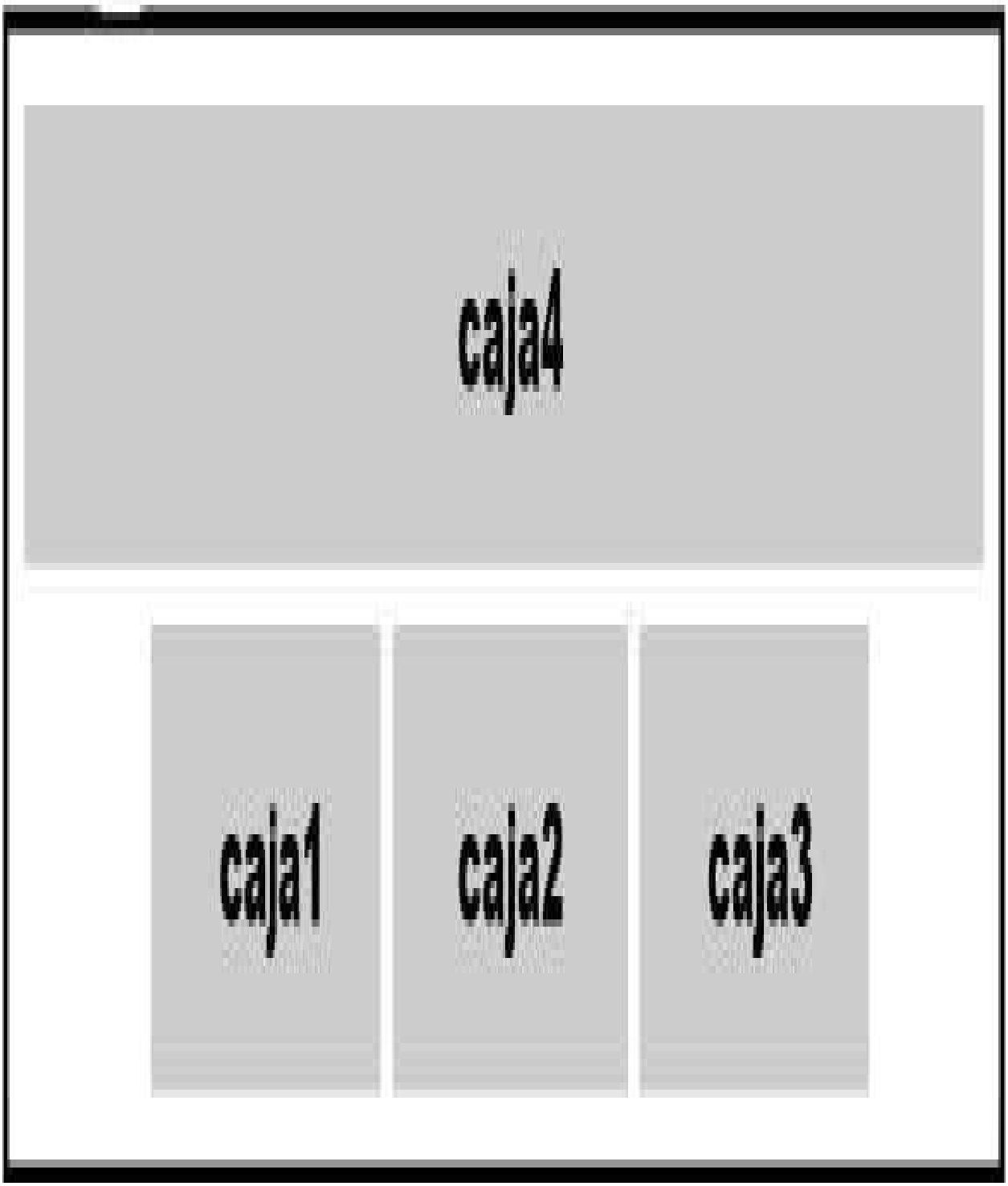


Figura 2-23

Usar flex-wrap: wrap-reverse para crear una nueva línea.

2.5.12 align-content

Cuando un contenedor flexible tiene múltiples líneas, es posible que haga falta alinearlas. La propiedad align-content alinea las líneas dentro de un contenedor flexible y funciona como align-items pero para múltiples líneas.

```
#contenedor {
   display: flex;
   flex-wrap: wrap;
   align-content: flex-start;
   width: 600px;
   height: 200px;
}
#caja1{
   flex: 1 1 100px;
}
#caja2{
   flex: 1 1 100px;
}
#caja3{
   flex: 1 1 100px;
}
#caja4{
   flex: 1 1 400px;
}
```

Código 2-55

Alineación de varias líneas con la propiedad align-content.

Para poder alinear las líneas tiene que haber espacio libre disponible y por eso en el Código 2-55 añadimos la propiedad height para el contenedor. Cada caja se declara como flexible con un tamaño inicial y el elemento contenedor se define como un contenedor multilínea con la propiedad flex-wrap. Esto crea un contenedor flexible con dos líneas similares al ejemplo del Código 2-54, pero con un espacio vertical libre para jugar.

La propiedad align-content puede tomar seis valores: flex-start, flex-end, center, space-between, space-around y stretch. De nuevo el último valor es el aplicado de forma predeterminada y amplía las líneas para llenar el espacio disponible, salvo que sea declarado un tamaño fijo para los elementos.

Vamos a ver cómo funcionan los diferentes valores:

caja contenedora

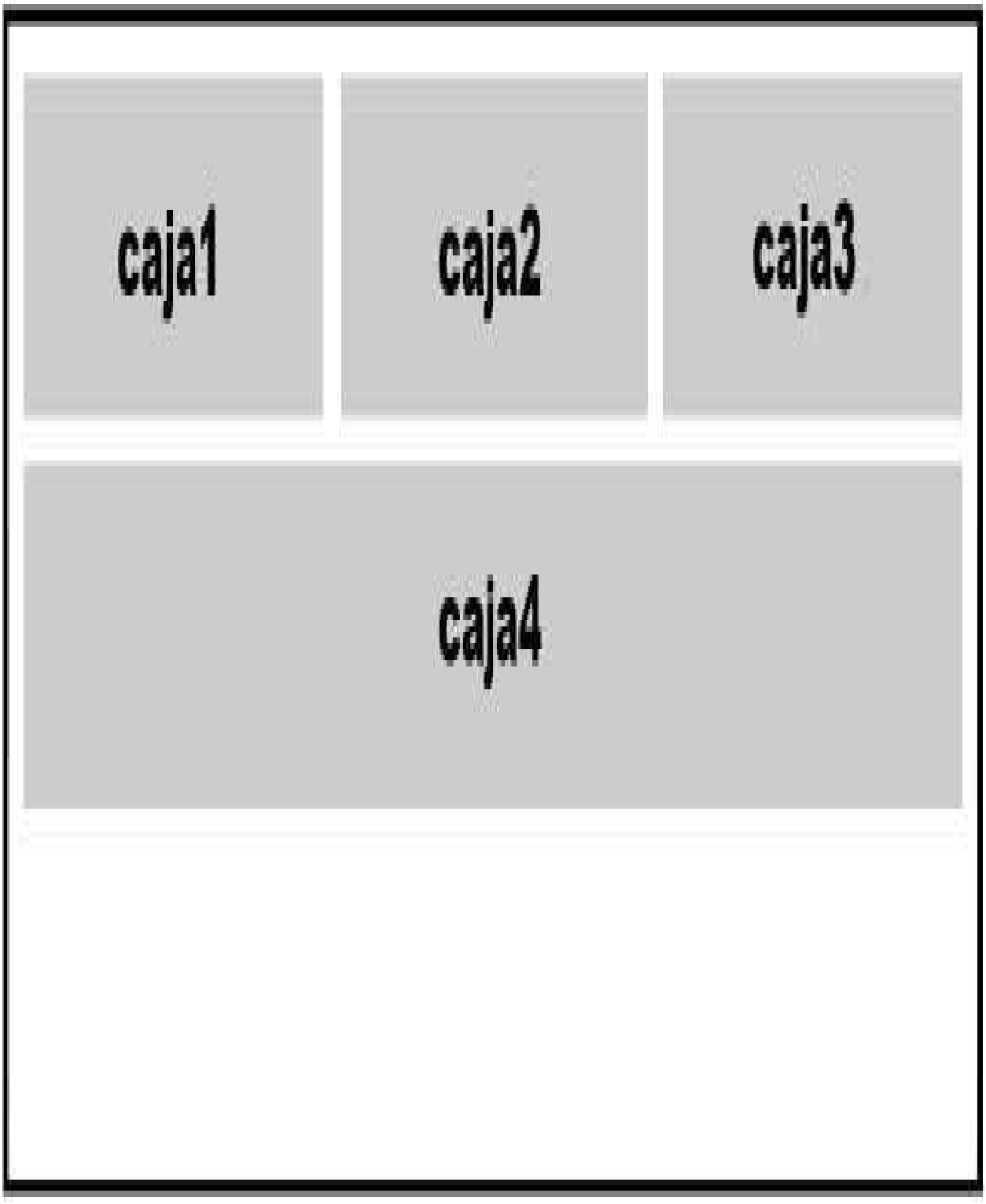


Figura 2-24

Alineación de líneas con align-content: flex-start.

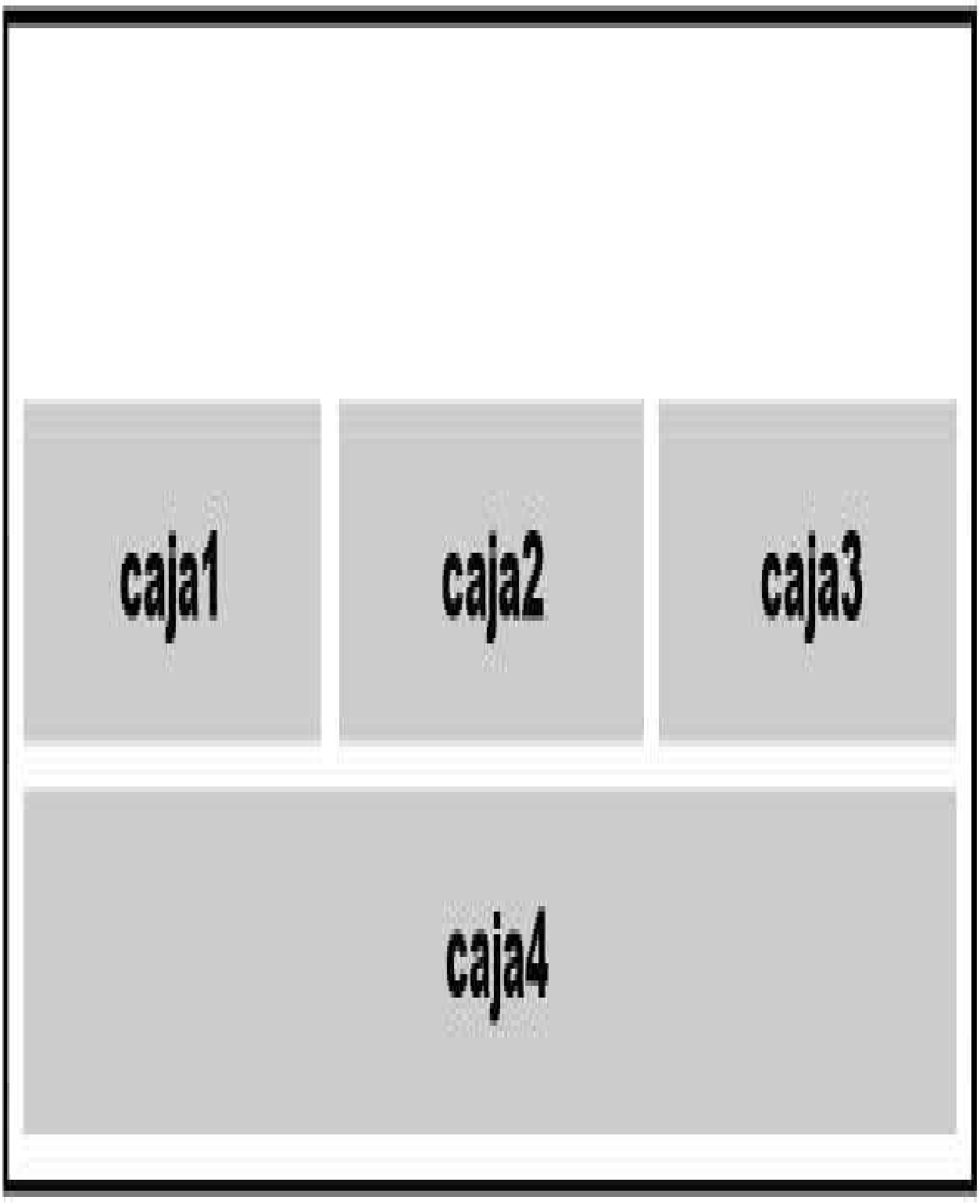

caja contenedora
caja1
caja2
caja3
caja4

Figura 2-25

Alineación de líneas con align-content: flex-end.

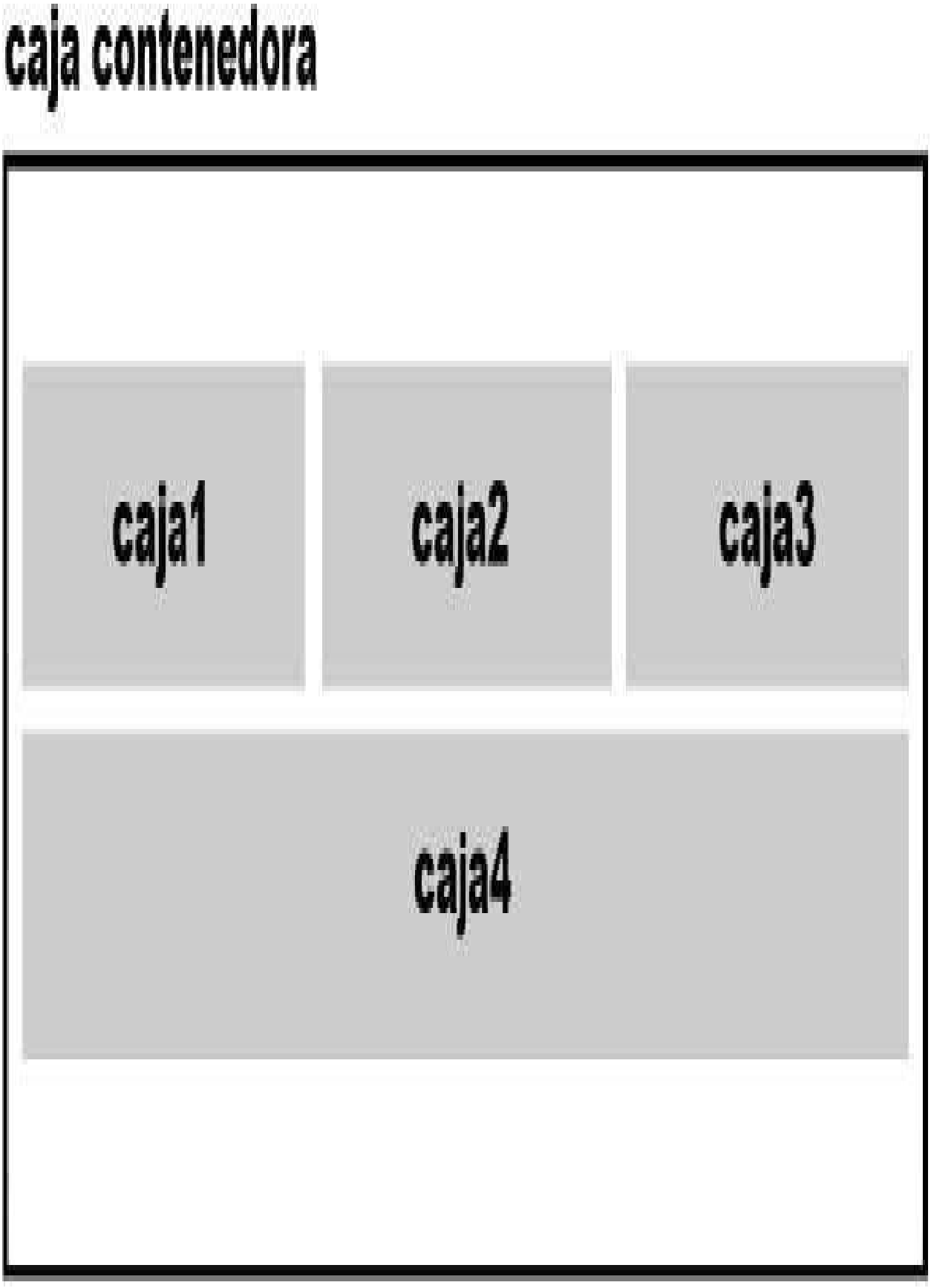

caja contenedora
caja1
caja2
caja3
caja4

Figura 2-26

Alineación de líneas con align-content: center.

caja contenedora

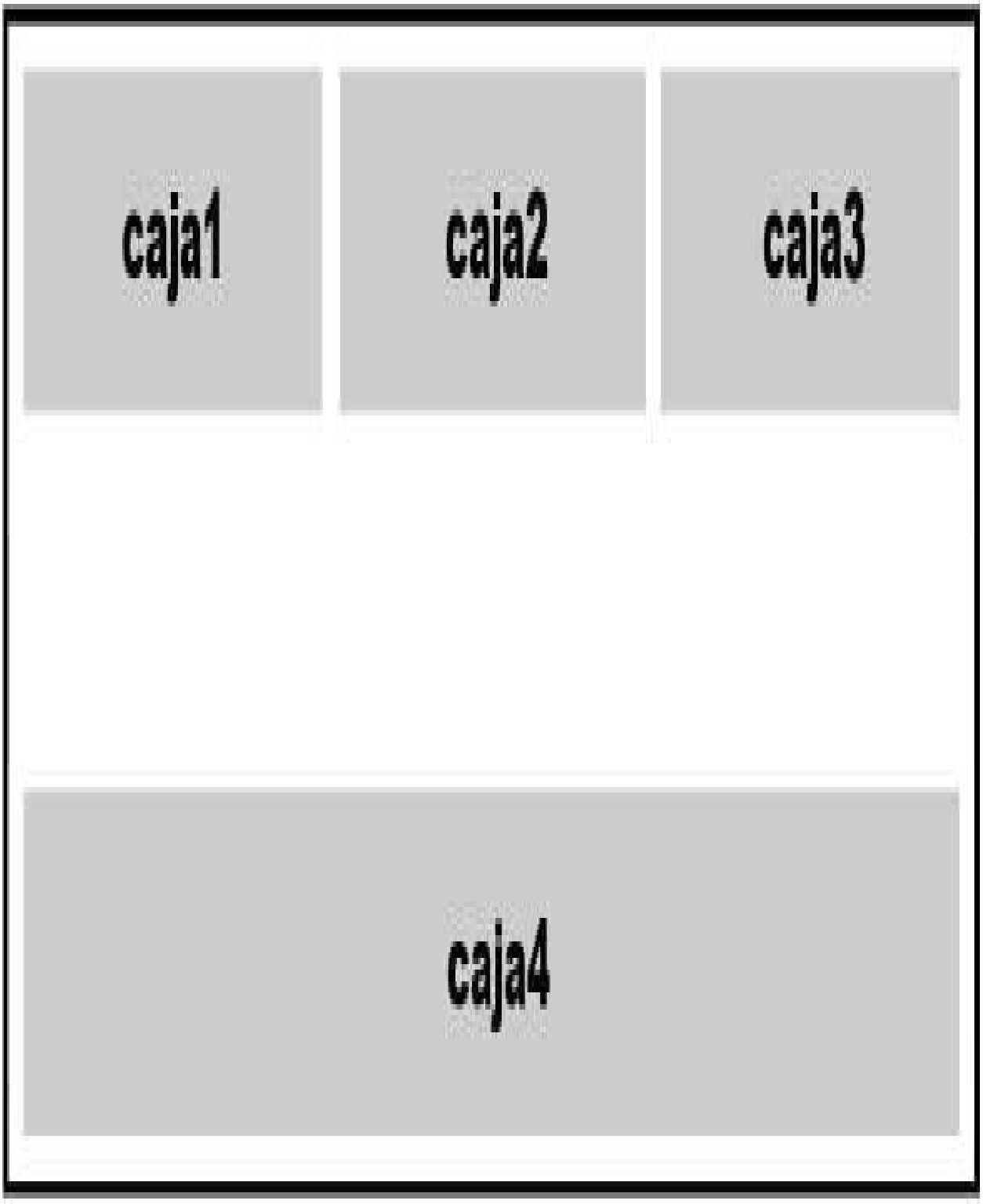


Figura 2-27

Alineación de líneas con align-content: space-between.

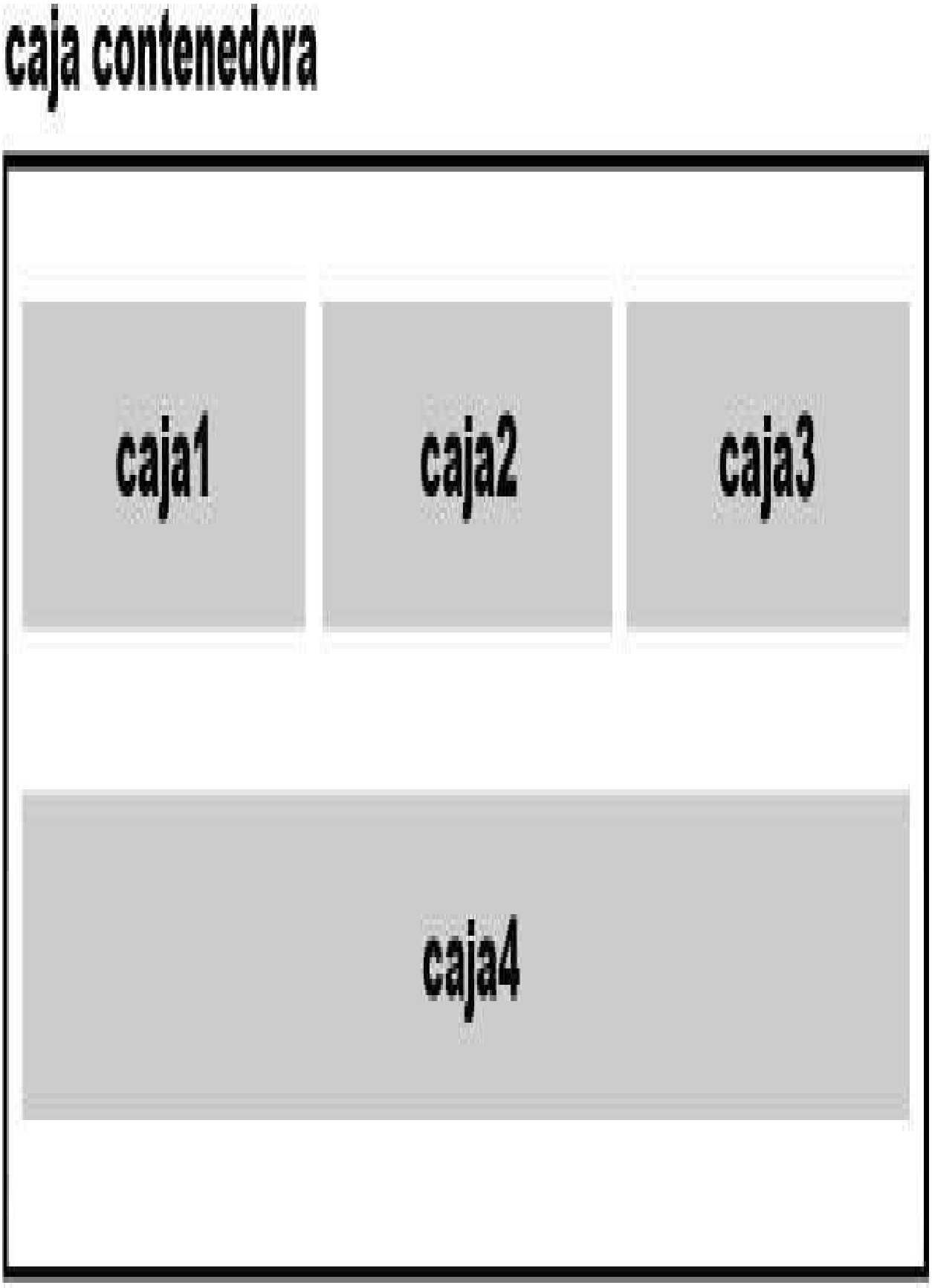

caja contenedora
caja1
caja2
caja3
caja4

Figura 2-28

Alineación de líneas con align-content: space-around.

caja contenedora

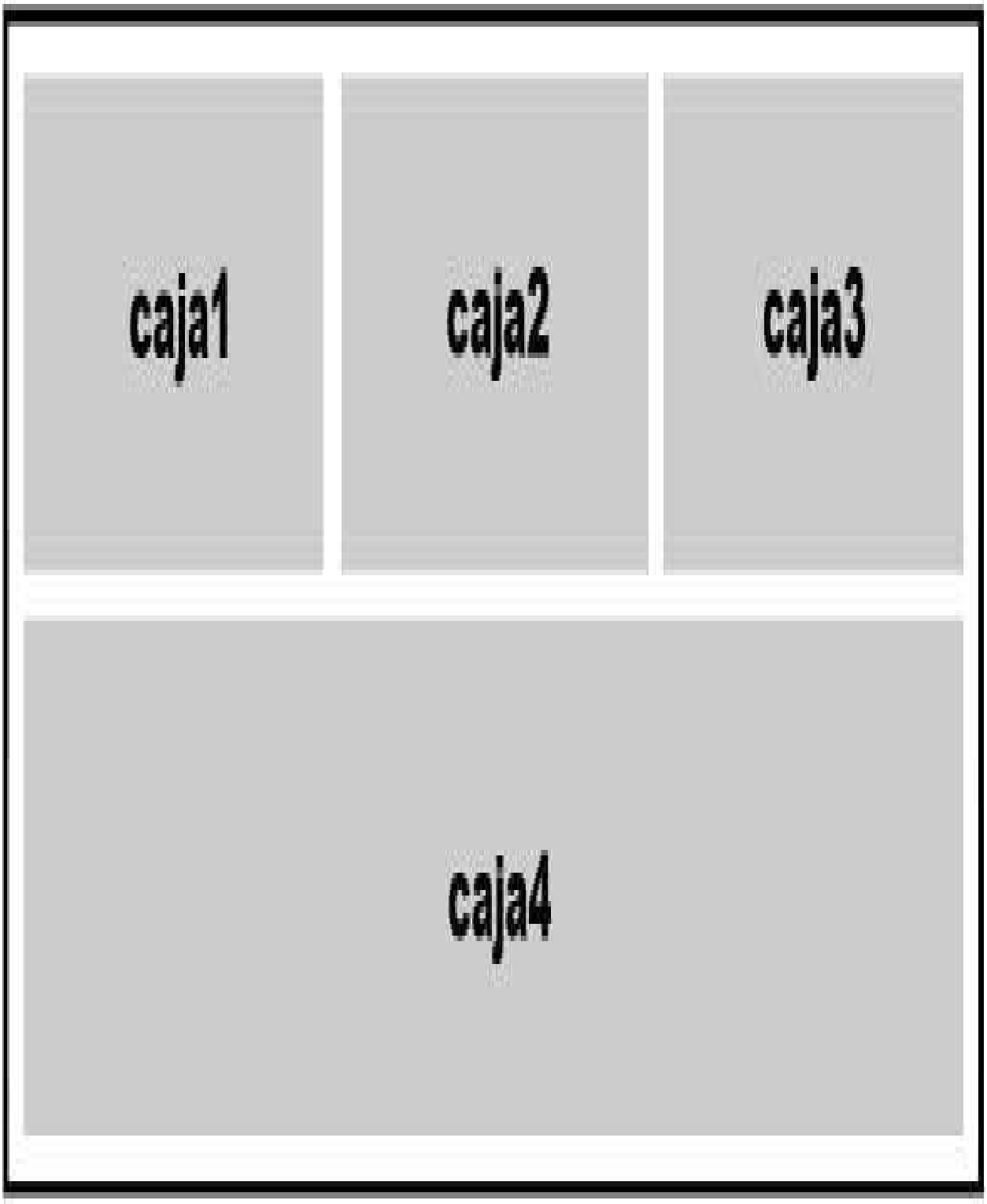


Figura 2-29

Alineación de líneas con align-content: stretch.

Hágalo usted mismo

Pruebe los diferentes valores disponibles para la propiedad align-content utilizando las reglas CSS del Código 2-55. Recuerde de nuevo agregar otras propiedades a los elementos, como fondo de color, altura, o un borde, para ver e identificar cada caja en la pantalla.

Importante

Para el momento de preparar este manual, el único navegador capaz de trabajar con el Modelo de caja flexible es Google Chrome y por tanto solo podrá probar estos ejemplos en este navegador y usando siempre el prefijo -webkit-(por ejemplo, -webkit-align-content: center;).

* Nota del traductor: En inglés se utiliza una analogía con un árbol familiar para referirse a los elementos anidados: De este modo podríamos decir que los elementos son hermanos (siblings) y que cada uno de ellos es hijo (child) de .

3. Propiedades CSS3

3.1 Las nuevas reglas

La Web cambió para siempre cuando a principios de este siglo nuevas aplicaciones desarrolladas a través de implementaciones de Ajax mejoraron el diseño y la experiencia de los usuarios. La Web 2.0, que es como se llamó al nuevo nivel de desarrollo, representó un cambio no solo en la forma en la que se transmite la información, sino también en cómo se diseñan los sitios web y las aplicaciones.

Los códigos implementados en esta nueva generación de sitios web pronto se convirtieron en el estándar. La innovación se hizo tan importante para el éxito de cualquier cosa en Internet que los programadores desarrollaron bibliotecas enteras de Javascript para superar las limitaciones y satisfacer las necesidades de los diseñadores. La falta de soporte por parte de los navegadores era evidente, pero el W3C (World Wide Web Consortium), organismo encargado de los estándares web, no prestó la atención necesaria al mercado y trató de seguir su propio camino. Afortunadamente, algunos brillantes programadores continuaron desarrollando nuevos estándares y pronto nació el lenguaje HTML5. Cuando volvió la calma, la integración de HTML, CSS y Javascript amparados por HTML5 estaban al nivel de un caballero victorioso y valiente que fue capaz de dirigir sus tropas hacia el palacio enemigo.

A pesar de que la agitación es reciente, la batalla comenzó hace mucho con la primera especificación de la tercera versión de CSS. Cuando en 2005 esta tecnología pasó a ser considerada un estándar oficial, CSS estaba listo para proporcionar las características exigidas por los desarrolladores, creadas durante años por los programadores utilizando códigos Javascript muy complicados y no siempre compatibles.

En este capítulo vamos a estudiar las contribuciones de CSS3 para HTML5 y todas las nuevas propiedades que simplifican la vida de los diseñadores y programadores.

3.1.1 CSS3 ha enloquecido

Originalmente CSS se utilizaba solo para definir la apariencia y el formato, pero ya no. Con el objetivo de reducir el uso de código Javascript y estandarizar las características más apreciadas, CSS3 no se ocupa solo de diseño y estilos web, sino también de la forma y el movimiento. La especificación CSS3 se presenta en módulos que proporcionan una especificación estándar para todos los aspectos implicados en la presentación visual del documento. Desde esquinas redondeadas y sombras hasta transformaciones y reorganización de los elementos, todos los efectos posibles de los aplicados previamente usando Javascript han sido incluidos. Ante cambios de esta envergadura, CSS3 se convirtió en una tecnología casi completamente nueva frente a las versiones anteriores.

3.1.2 Documento HTML

Las nuevas propiedades de CSS3 son extremadamente poderosas y deben ser estudiadas de forma individual, pero para hacer más fácil el proceso vamos a aplicar todas ellas al mismo documento HTML, así que para comenzar establezcamos el código HTML con el que trabajaremos, además de algunos estilos CSS que utilizaremos como base para los próximos ejercicios:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Nuevos estilos CSS3</title>
  <link rel="stylesheet" href="nuevoscss3.css">
</head>
<body>
  <header id="contenedor">
    <span id="título">Estilos CSS3 Web2.0</span>
  </header>
</body>
</html>
```

Código 3-1

Un documento sencillo para probar nuevas propiedades.

Nuestro documento solo tiene una caja con un breve texto en el interior. El elemento <header> utilizado para este cuadro podría ser sustituido por , <nav>, <sección> o cualquier otro elemento estructural de acuerdo con la ubicación en el diseño y su función. Después de aplicar estilos, la caja del Código 3-1 se verá como un encabezado, de ahí que hayamos usado el elemento <header>.

Dado que el elemento <font> está desfasado en HTML5, como ya hemos explicado antes, los elementos que utilizados habitualmente para mostrar textos son para líneas cortas y <p> para párrafos. Por esta razón, el texto de nuestro documento está insertado entre etiquetas .

Hágalo usted mismo

Utilice el Código 3-1 para crear el documento HTML con el que trabajará en este capítulo. También tendrá que crear un nuevo archivo CSS llamado nuevoscss3.css para almacenar los estilos CSS.

Los estilos básicos de nuestro documento serán los siguientes:

```
body {
  text-align: center;
}
#contenedor {
  display: block;
  width: 500px;
  margin: 50px auto;
  padding: 15px;
  border: 1px solid #999999;
  background: #FFFFFF;
}
#título {
  font: bold 36px verdana, sans-serif;
```

Código 3-2

Reglas CSS básicas para este capítulo.

No hay nada nuevo en las reglas del Código 3-2. Son solo los estilos necesarios para dar forma a nuestra cabecera y crear una caja larga, situada en el centro de la ventana, con un fondo gris y un borde de 1 px, además de un texto de gran tamaño dentro de la misma que dice Estilos CSS3 Web 2.0.

Estilos CSS3 Web2.0

Figura 3-1

Cabecera con estilos tradicionales.

3.1.3 border-radius

Durante muchos años los desarrolladores han sufrido tratando de obtener hermosas esquinas para las cajas en sus páginas web: se trataba generalmente de un proceso agotador. Si ve cualquier vídeo de las primeras presentaciones de las características incorporadas en HTML5, notará que cada vez que alguien hablaba de la propiedad CSS que permite generar fácilmente esquinas redondeadas, el público enloquecía. Ésta era una de esas cosas que uno podía esperar que fueran muy sencillas de implementar y, sin embargo, durante años no fue una tarea fácil. Es por eso que, entre todas las nuevas y sorprendentes propiedades incorporadas en CSS3, la primera que vamos a explorar es border-radius:

```
body {
  text-align: center;
}
#contenedor {
  display: block;
  width: 500px;
  margin: 50px auto;
  padding: 15px;
  border: 1px solid #999999;
  background: #FFFFFF;
    border-radius: 20px;
}
#título {
  font: bold 36px verdana, sans-serif;
}
```

Código 3-3

Generación de esquinas redondeadas.

Estilos CSS3 web2.0

Figura 3-2

Esquinas redondeadas.

Si cada esquina tiene el mismo valor, podemos declarar solo un valor para esta propiedad. Sin embargo, tal como ocurre con las propiedades margin y padding, se puede seleccionar un valor diferente para cada esquina.

```
body {
  text-align: center;
}
#contenedor {
  display: block;
  width: 500px;
  margin: 50px auto;
  padding: 15px;
  border: 1px solid #999999;
  background: #FFFFFF;
    border-radius: 20px 10px 30px 50px;
}
#título {
  font: bold 36px verdana, sans-serif;
}
```

Código 3-4

Declaración de valores diferentes para cada rincón.

Como se puede ver en el Código 3-4, los cuatro valores asignados a la propiedad border-radius representan cuatro puntos diferentes. Sepa que se ubican en dirección horaria: esquina superior izquierda, esquina superior derecha, esquina inferior derecha y esquina inferior izquierda y siempre debe respetarse este orden.

Hágalo usted mismo

En algunos navegadores de estas nuevas propiedades son aún experimentales. Si no puede ver el efecto que se supone que produce en la pantalla, intente agregar los prefijos correspondientes para el navegador que está utilizando (por ejemplo, -moz-,-webkit-, etc), tal como lo hicimos antes con las propiedades estudiadas en el Capítulo 2.

CSS Styles Web 2.0

Figura 3-3

Valores diferentes para cada esquina.

Al igual que con margin o padding, border-radius también puede funcionar con dos valores. En ese caso el primer valor se asigna a las esquinas primera y tercera, de nuevo en sentido horario, (arriba a la izquierda, abajo a la derecha) y el segundo a las esquinas segunda y cuarta en el mismo sentido (arriba a la derecha y abajo a la izquierda). También es posible modificar las curvas, proporcionando nuevos valores separados por una barra inclinada (/). Los valores a la izquierda de la barra representan el radio horizontal y los valores a la derecha, el radio vertical: la combinación de estos valores genera una elipse.

```
body {
  text-align: center;
}
#contenedor {
  display: block;
  width: 500px;
  margin: 50px auto;
  padding: 15px;

  border: 1px solid #999999;
  background: #FFFFFF;
    border-radius: 20px / 10px;
}
#título {
  font: bold 36px verdana, sans-serif;
}
```

Código 3-5

Generación de curvas elípticas.

CSS Styles Web 2.0

Figura 3-4

Curvas elípticas.

3.1.4 box-shadow

Ahora que finalmente tenemos unas esquinas atractivas, podemos ir más allá. Las sombras son otro efecto que antes resultaba muy complicado de conseguir en CSS. Durante años, los diseñadores necesitaban combinar imágenes, elementos y algunas propiedades CSS para generar sombras. Gracias a CSS3 y a la nueva característica box-shadow ahora es posible aplicar una sombra a un elemento solo con una línea de código:

Hágalo usted mismo

Copie los estilos que desea probar en el archivo CSS llamado nuevoscss3. css y abra el archivo HTML con el Código 3-1 en su navegador. Cambie el valor de la propiedad border-radius para comprender su efecto.

```
body {
  text-align: center;
}
#contenedor {
  display: block;
  width: 500px;
  margin: 50px auto;
  padding: 15px;
  border: 1px solid #999999;
  background: #FFFFFF;

  box-shadow: rgb(150,150,150) 5px 5px;
}
#título {
  font: bold 36px verdana, sans-serif;
```

Código 3-6

Aplicación de una sombra a la caja.

La propiedad box-shadow necesita por lo menos tres valores. El primero de ellos, que se puede ver en la regla en el Código 3-6, es el color y lo hemos establecido utilizando rgb() con números decimales, pero se puede escribir en hexadecimal, como lo hicimos antes para otras propiedades. Los dos siguientes valores, expresados en píxeles, ajustan el desplazamiento de la sombra, que puede ser positivo o negativo e indican la distancia horizontal y vertical, respectivamente, entre la sombra y el elemento. Los valores negativos ubican la sombra a la izquierda y encima del elemento, mientras que los valores positivos crean una sombra a la derecha y debajo del elemento. Un valor igual a 0 posiciona la sombra detrás del elemento, ofreciendo la posibilidad de generar un efecto de desenfoque a su alrededor, como veremos a continuación.

CSS Styles Web 2.0

Figura 3-5

Sombra básica.

Hágalo usted mismo

Para probar los diferentes parámetros y posibilidades de creación de sombras, copie el Código 3.6 en el archivo CSS y abra en su navegador el documento HTML que contiene el Código 3-1. Experimente con los valores de la propiedad box-shadow y utilice el mismo código para los nuevos parámetros que se estudian a continuación.

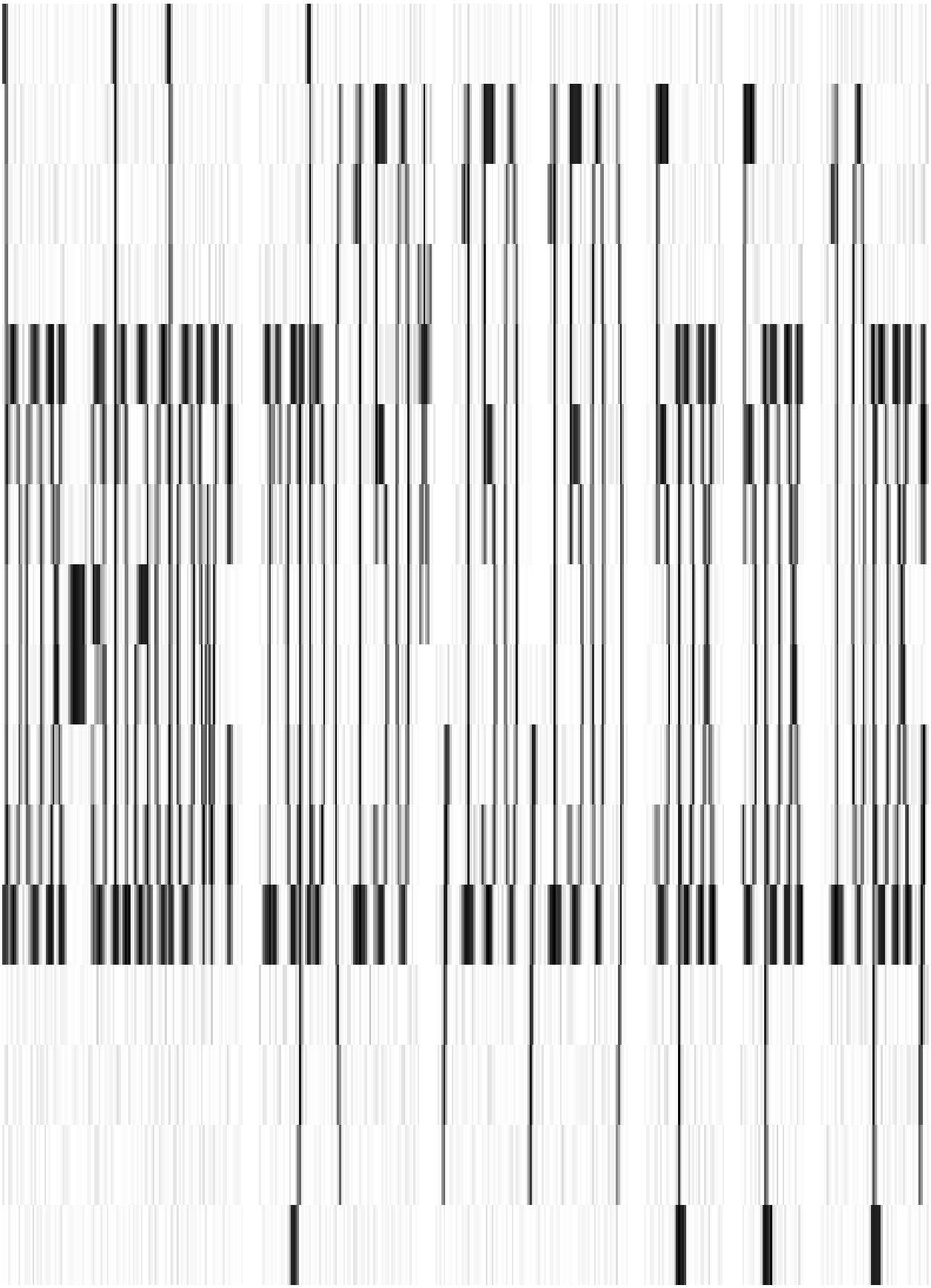
Importante

Tenga en cuenta que actualmente estas propiedades son experimentales en algunos navegadores. Para usarlas, añada al declararlas los prefijos -moz- o -webkit - en función del navegador (Mozilla Firefox o Google Chrome para los prefijos indicados).

La sombra que tenemos de momento es sólida, sin degradado ni transparencia, así que no tiene un verdadero aspecto de sombra, así que aún quedan algunos parámetros y algunos cambios que podemos aplicar para mejorar la apariencia

de la sombra.

Podemos añadir un cuarto valor a la propiedad box-shadow para establecer la distancia del desenfoque y hacer que la sombra tenga una apariencia más real. Puede probar este parámetro declarando un valor de 10 px en la regla adecuada del Código 3-6, como en el siguiente ejemplo:



Código 3-7

Agregar el valor de desenfoque a box-shadow.

CSS Styles Web 2.0

Figura 3-6

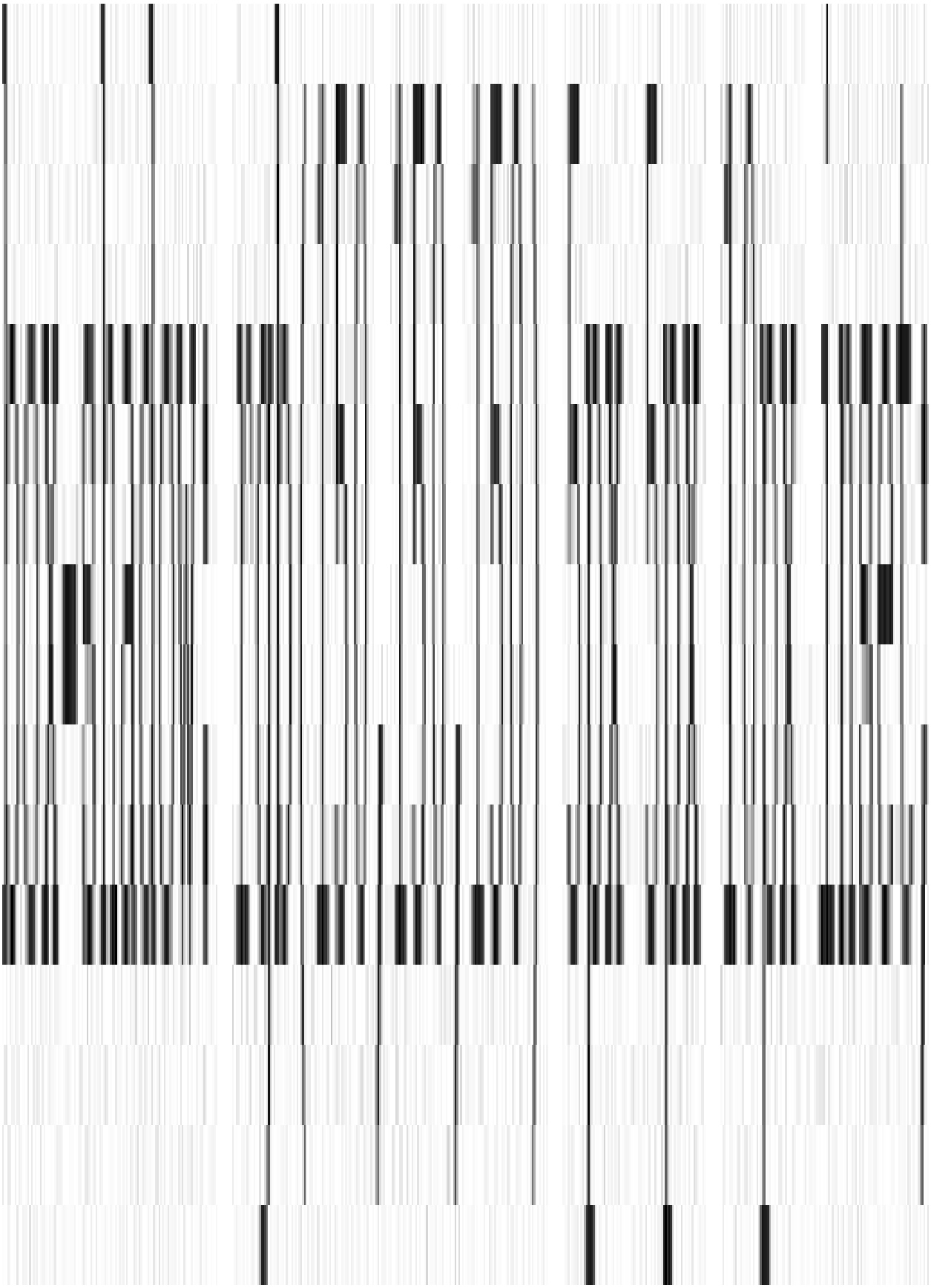
Sombra real.

Si añade otro valor en px al final de la propiedad se propagará la sombra. Se trata de un efecto que cambia ligeramente el aspecto de la sombra ampliando la zona que ésta cubre.

Hágalo usted mismo

Agregue un valor de 20 px al final del estilo del Código 3-7. Combine este código con el Código 6.3 y compruebe el resultado.

El último valor posible para box-shadow no es un número sino la palabra clave inset. Esta palabra clave convierte la sombra externa en una sombra interior que proporciona un efecto de profundidad a la caja.



Código 3-8

Crear una sombra interior.

El estilo del Código 3-8 genera una sombra interna de 5 px desde el borde de la caja y un efecto de desenfoque de 10 px, como se muestra en la siguiente figura.

CSS Styles Web 2.0

Figura 3-7

Sombra interior.

Hágalo usted mismo

Los estilos del Código 3-7 y 3-8 son solo ejemplos. Para comprobar el efecto en su navegador, debe aplicar los cambios en las reglas del Código 3-6.

Importante

Las sombras no expanden el elemento ni aumentan su tamaño, así que deberá comprobar que hay suficiente espacio disponible para que se vea la sombra.

3.1.5 text-shadow

Ahora que sabemos todo acerca de las sombras, es posible que desee generar una para cada elemento del documento. La propiedad box-shadow se ha diseñado específicamente para las cajas. Si intenta aplicar este efecto a un elemento , por ejemplo, la sombra no será aplicada a su contenido sino a la caja invisible que ocupa este elemento en pantalla. Por tanto, para aplicar sombras a las formas irregulares de los textos, existe una propiedad especial que se llama text-shadow. Veamos cómo funciona:

```
body {
  text-align: center;
}
#contenedor {
  display: block;
  width: 500px;
  margin: 50px auto;
  padding: 15px;
  border: 1px solid #999999;
  background: #FFFFFF;
}
#título {
  font: bold 36px verdana, sans-serif;
  text-shadow: rgb(150, 150, 150) 3px 3px 5px;
}
```

Código 3-9

Añadir una sombra al título.

Los valores para de text-shadow son similares a los de box-shadow. Permite definir el color de la sombra, la distancia horizontal desde la sombra hasta al objeto, la distancia vertical y el radio de desenfoque.

En el Código 3-9 se aplica una sombra a la cabecera con una distancia de solo 3 px y con un radio de desenfoque de 5, como se muestra en la siguiente figura:

CSS Styles Web 2.0

Figura 3-8

Sombra de texto.

3.1.6 @font-face

Una sombra de texto es un truco muy interesante y difícil de conseguir con los métodos anteriores, pero solo proporciona un efecto tridimensional sin cambiar realmente el texto. Aplicar una sombra es como pintar un coche viejo, seguirá siendo el mismo coche. En el caso del texto, será la misma fuente. Por otra parte, el problema con las fuentes es tan antiguo como la Web. El usuario típico de la red suele tener un número limitado de fuentes instaladas, no siempre se trata de las mismas familias de fuentes, de manera que algunos tienen un tipo de letra que otros no. Durante años, los sitios web solo podían utilizar un conjunto mínimo de fuentes, que la mayoría de usuarios tiene, para procesar y mostrar la información en la pantalla.

La función @font-face permite a los diseñadores utilizar cualquier tipo de letra sus sitios web ya que hace posible incluir un archivo de fuentes.

```
body {
  text-align: center;
}
#contenedor {
  display: block;
  width: 500px;
  margin: 50px auto;
  padding: 15px;
  border: 1px solid #999999;
  background: #FFFFFF;
}
#title {
  font: bold 36px MiFuente, verdana, sans-serif;
}
@font-face {
  font-family: 'MiFuente';
  src: url('font.ttf');
}
```

Código 3-10

Nueva fuente para el título.

Hágalo usted mismo

Descargue el archivo font.ttf de nuestro sitio web o utilice un archivo de fuentes propio y ubíquelo en la misma carpeta o directorio de su archivo CSS (para descargar el archivo, vaya al www.minkbooks.com/content/). Puede conseguir más fuentes gratuitas como ésta en la siguiente dirección: www.moorstation.org/typoasis/designers/steffmann/.

Importante

El archivo con el tipo de letra debe estar en el mismo dominio que la página web (o en el mismo equipo, en este caso). Ésta es una restricción de algunos navegadores, como Mozilla Firefox.

La función @font-face necesita al menos dos propiedades para declarar la fuente y cargar el archivo. La propiedad font-family especifica el nombre que desea utilizar para hacer referencia a esa fuente en particular, y la propiedad src indica la URL del archivo con las especificaciones de la fuente. En el Código 3-10 se

asigna el nombre la MiFuente a la fuente y se indica que se encuentra en el archivo font.ttf.

Una vez que la fuente ha sido cargada, puede usarla en cualquier elemento del documento con solo escribir su nombre (MiFuente). En el estilo Font de la regla del Código 3-10 se especifica que el título se mostrará con la nueva fuente o con las alternativas verdana o sans-serif, si la fuente no se ha cargado de forma correcta.

CSS Styles Web 2.0

Figura 3-9

Fuente personalizada para el título.

3.1.7 linear-gradient

Los degradados son una de las características más atractivas incorporadas en CSS3. Con las técnicas anteriores era casi imposible implementarlos, pero ahora es realmente fácil hacerlo. La propiedad background, con unos pocos parámetros, es suficiente para convertir su documento en una página web de apariencia profesional.

Importante

Mozilla propuso una nueva sintaxis para degradados que aún no ha sido adoptada por los proveedores de otros navegadores. La nueva especificación incluye la palabra at para describir la posición de los degradados radiales. Consulte nuestro sitio web para obtener actualizaciones.

```
body {
  text-align: center;
}
#contenedor {
  display: block;
  width: 500px;
  margin: 50px auto;
  padding: 15px;
  border: 1px solid #999999;

  background: -webkit-linear-gradient(top, #FFFFFF, #666666);
  background: -moz-linear-gradient(top, #FFFFFF, #666666);
}
#título {
  font: bold 36px verdana, sans-serif;
}
```

Código 3-11

Creación de un degradado de fondo para la caja.

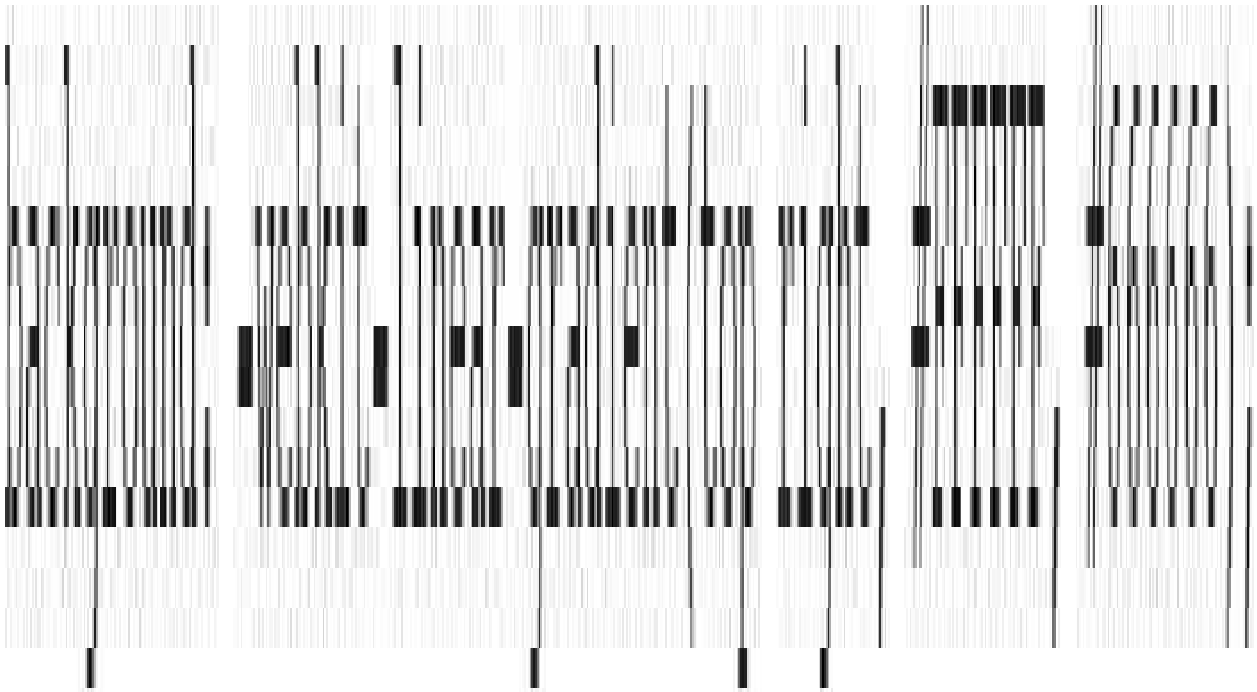
Los degradados se establecen como fondos, así que pueden aplicarse con la propiedad background o la propiedad background-image, más específica. La sintaxis para degradados lineales es linear-gradient(posición_inicial, desde-_color, hasta_color) y sus atributos indican el punto de partida y los colores utilizados para crear el degradado. El primer valor se declara con palabras clave top, bottom, left y right (tal como lo hicimos en el ejemplo del Código 3-11).

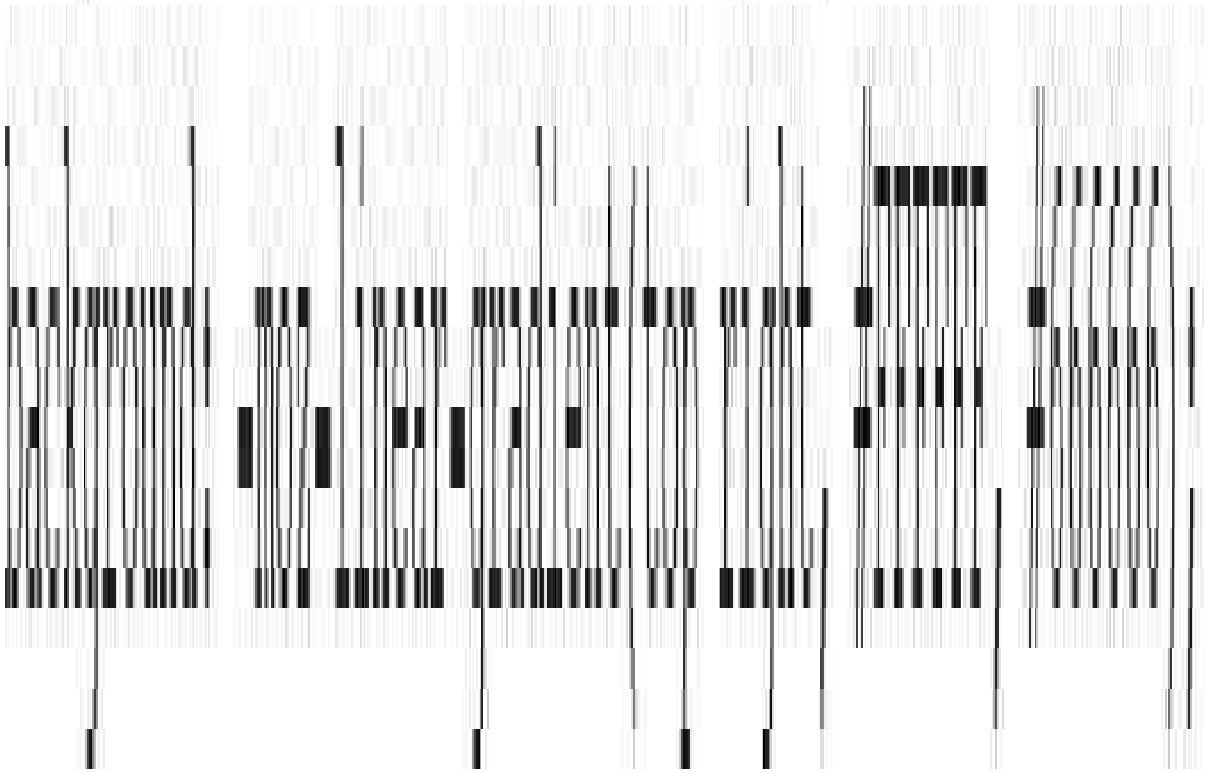
Estilos CSS3 Web2.0

Figura 3-10

Degradado lineal.

Las palabras clave también pueden combinarse para señalar una esquina del elemento, como en el ejemplo que viene a continuación.





Código 3-12

Combinación de palabras clave.

Estilos CSS3 Web2.0

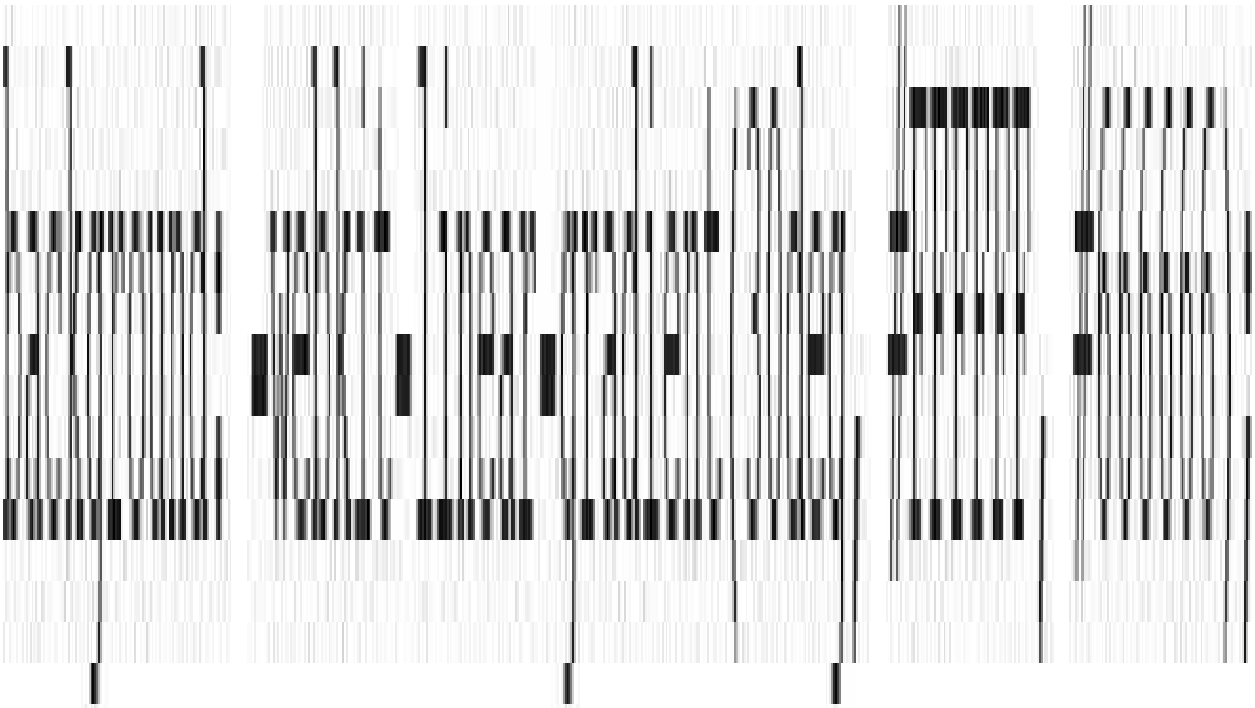
Figura 3-11

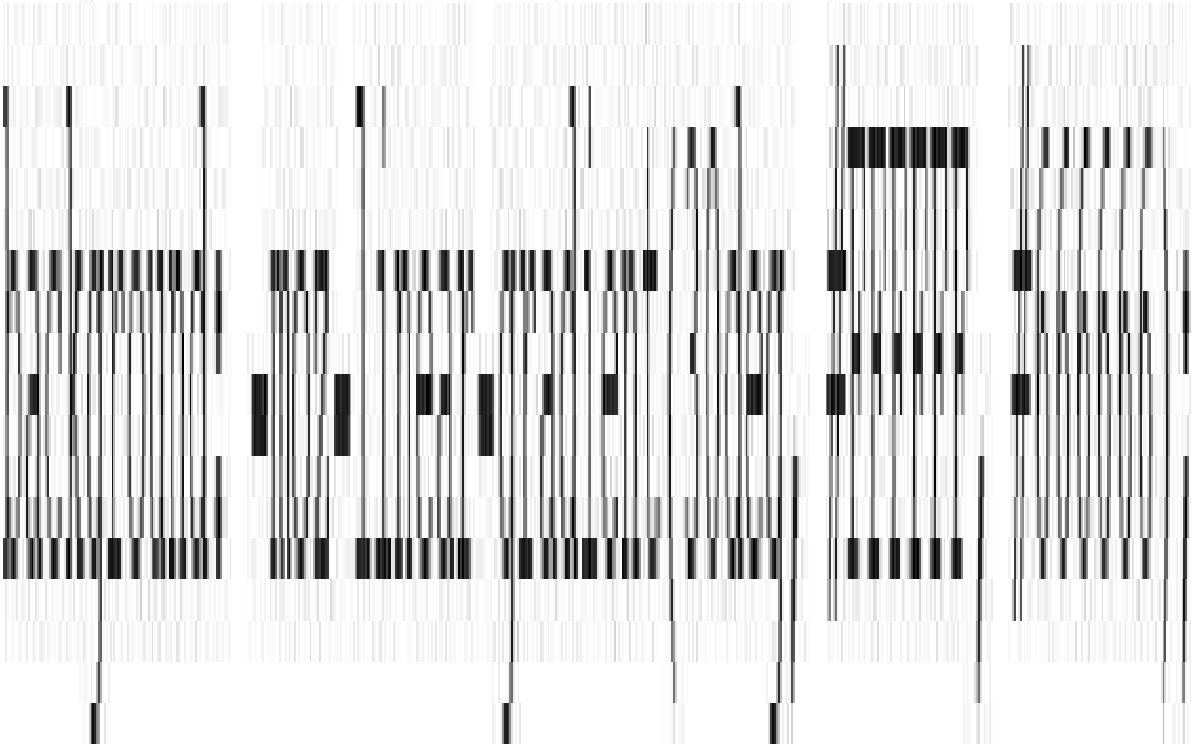
Posición de inicio diferente para un degradado lineal.

Importante

La nueva especificación declara las palabras clave con el prefijo to, que invierte la dirección del degradado (por ejemplo, to top), pero actualmente no todos los navegadores comprenden esta nueva sintaxis y no está claro si la sintaxis que se presenta en este capítulo se conservará o será abandonada. Para aprender más sobre la historia de la sintaxis de degradados, por favor visite nuestro sitio web y siga los enlaces de este capítulo.

La posición inicial puede ser también expresada por un ángulo:





Código 3-13

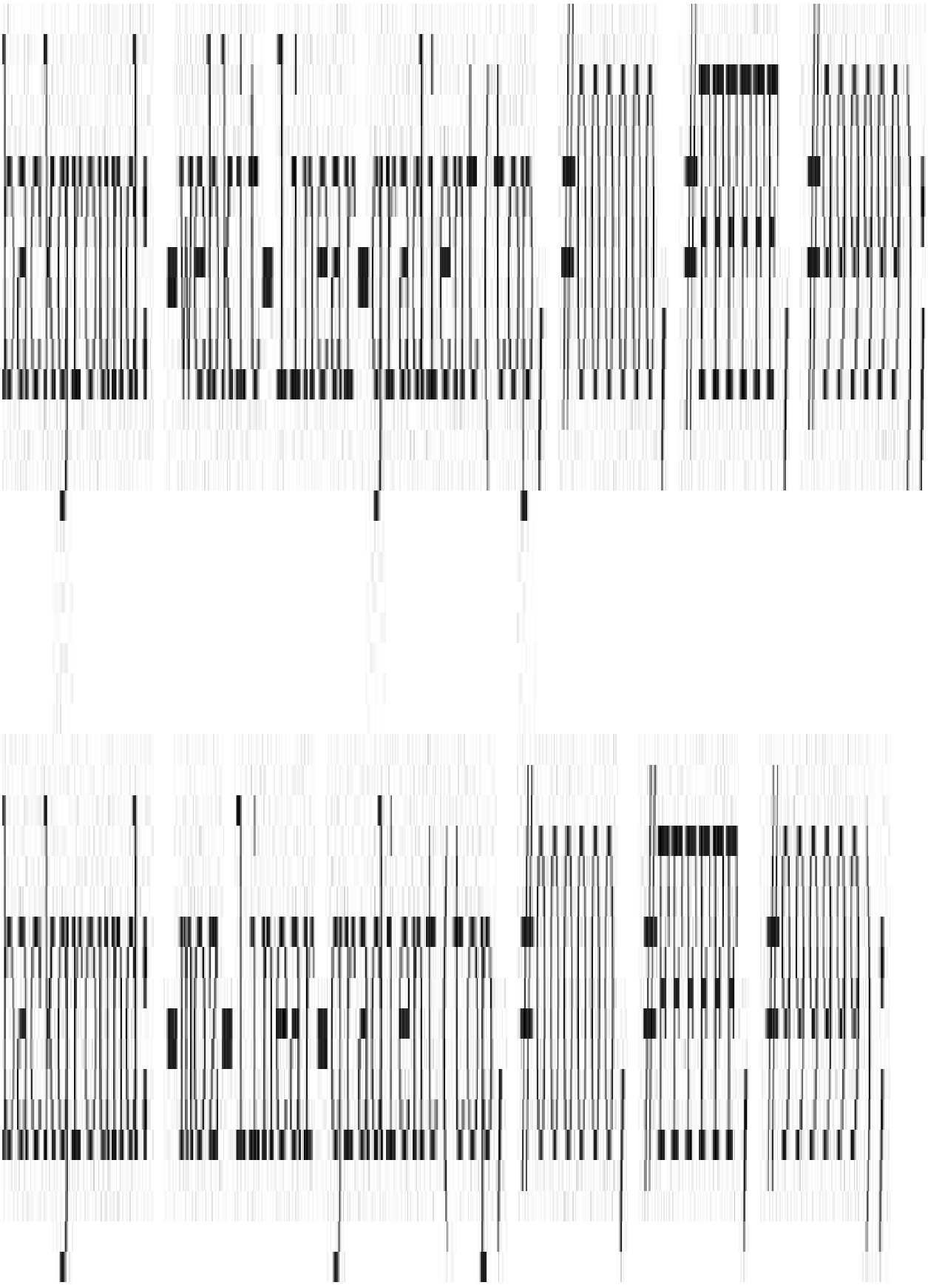
Creación de un degradado con una dirección de 30 grados.

Estilos CSS3 Web2.0

Figura 3-12

Posición inicial en grados para un degradado lineal.

Podemos añadir más valores para generar un degradado multicolor:



Código 3-14

Creación de un degradado multicolor.

CSS Styles Web 2.0

Figura 3-13

Degradado multicolor lineal.

El uso de la palabra clave transparent permite combinar el degradado con el fondo, pero sepa que este efecto también se puede lograr con la función rgba() que estudiaremos más adelante.

```
body {
  text-align: center;
  background: url(http://www.minkbooks.com/content/bricks2.jpg);
}
#contenedor {

  display: block;
  width: 500px;
  margin: 50px auto;
  padding: 15px;
  border: 1px solid #999999;

  background: -webkit-linear-gradient(top, transparent, #666666);
  background: -moz-linear-gradient(top, transparent, #666666);
}
#título {
  font: bold 36px verdana, sans-serif;
```

Código 3-15

Creación de un degradado transparente.

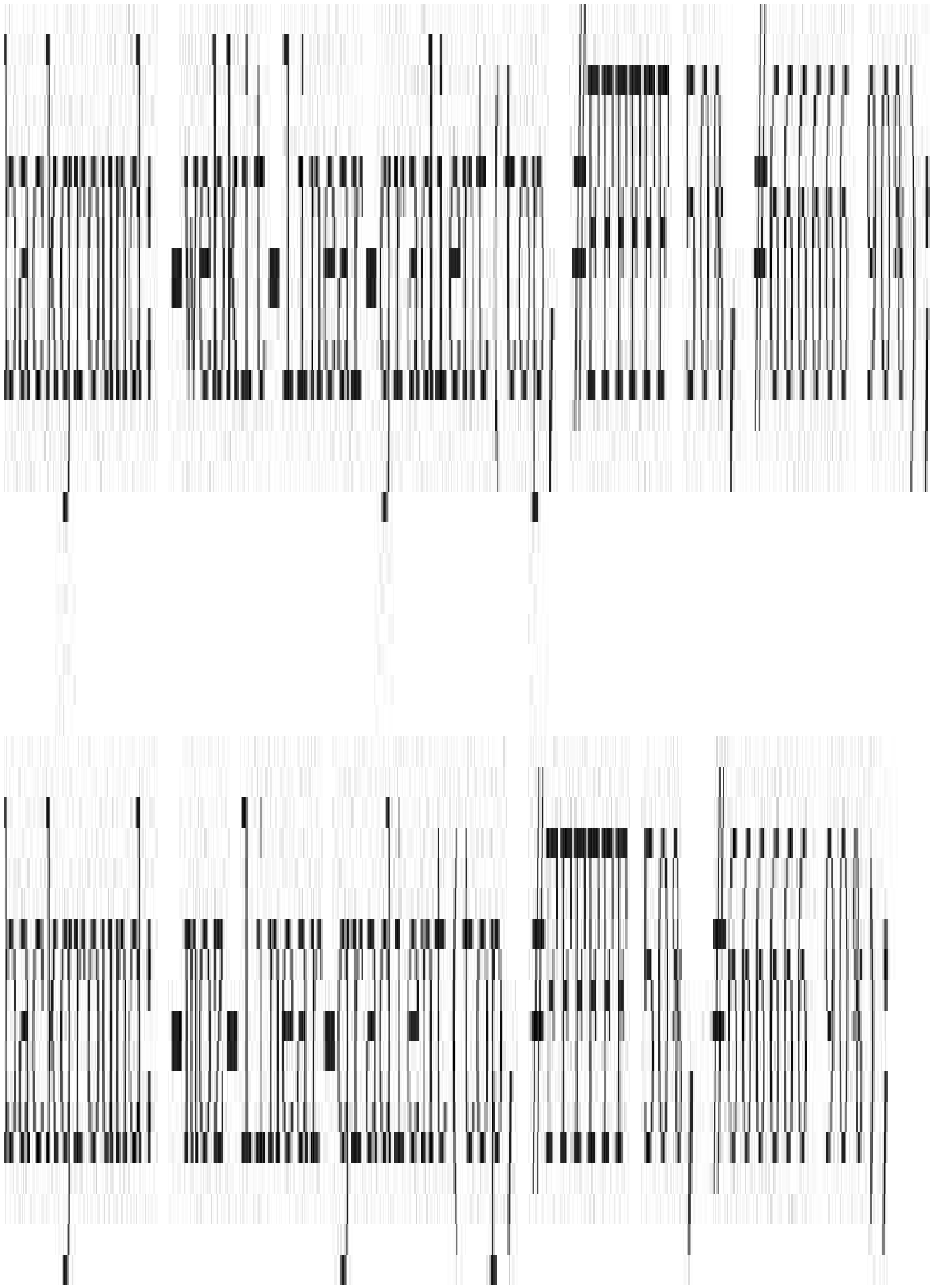
En el ejemplo del Código 3-15, añadimos una imagen de fondo para el elemento <body> que nos permitiera apreciar el efecto de la palabra clave transparent en pantalla. El resultado se muestra en la Figura 3-14.

CSS Styles Web 2.0

Figura 3-14

Degradado transparente.

Los parámetros de los colores se describen en la especificación con el nombre color-stop. Esto es porque al incluir un valor adicional se puede determinar el punto de inicio y de fin de cada color, y personalizar el degradado.



Código 3-16

Ajuste del parámetro color-stop.

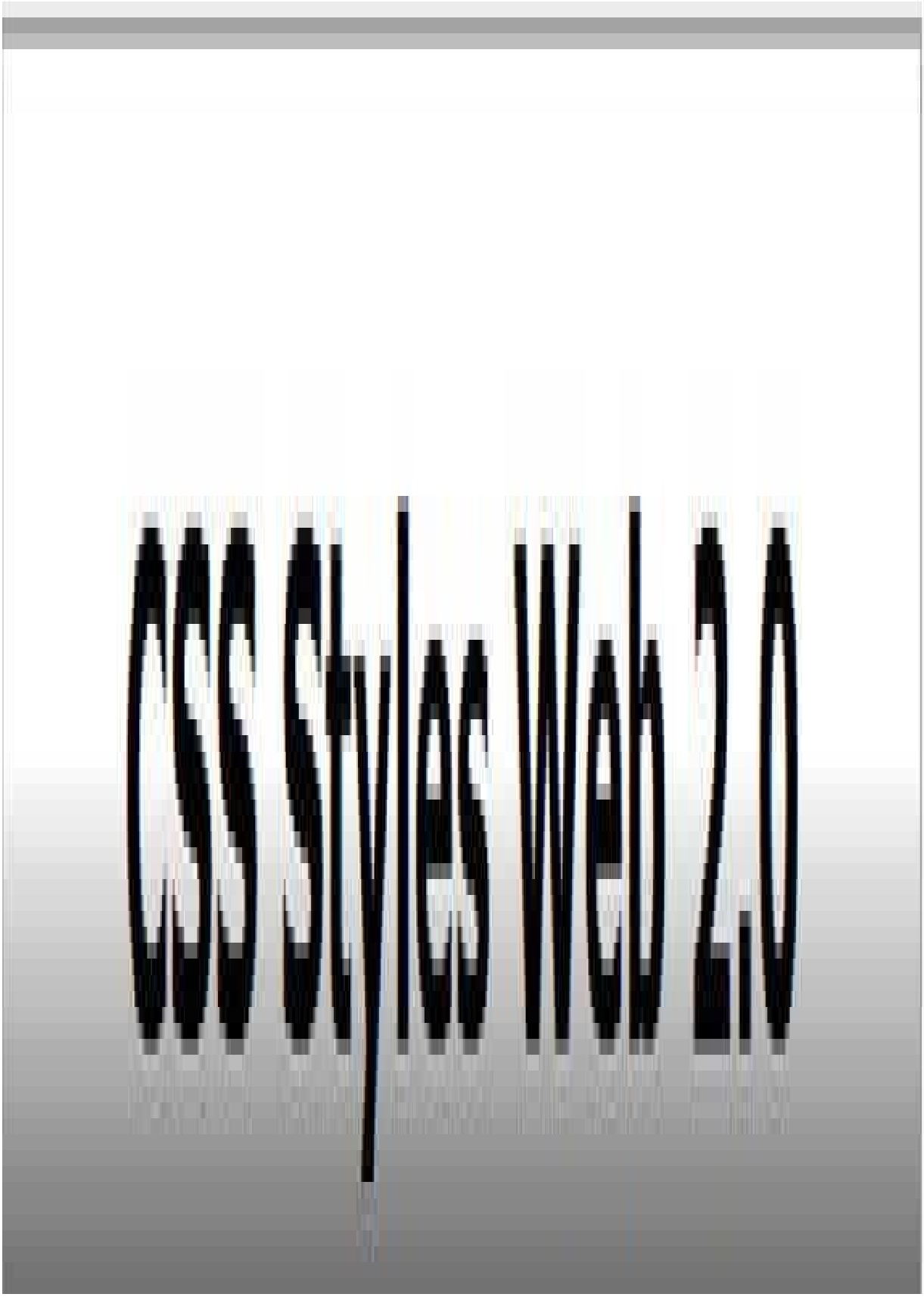
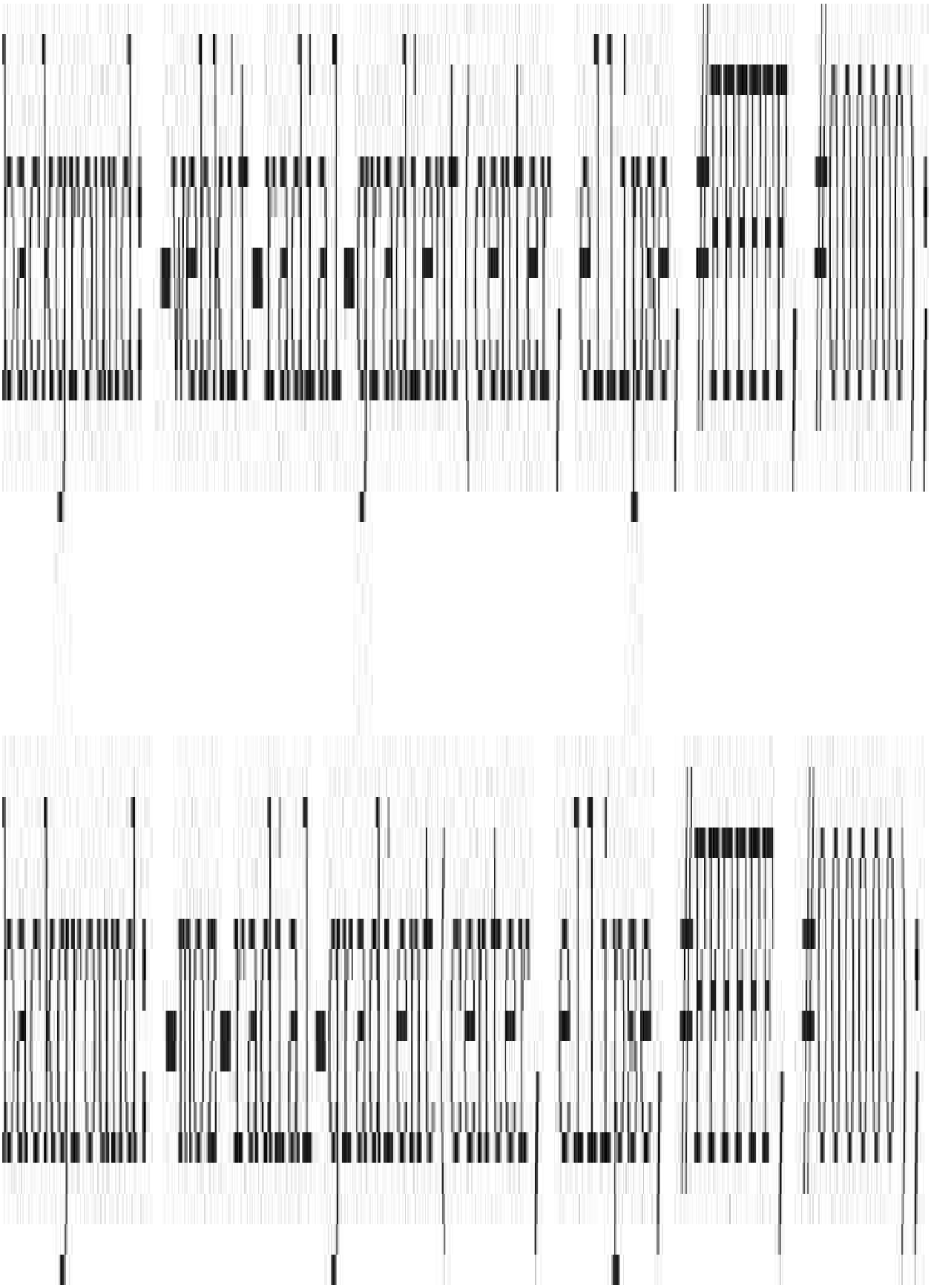

CSS Styles Web 2.0

Figura 3-15

Añadir valores color-stop.

3.1.8 radial-gradient

La sintaxis estándar actual para degradados radiales no es tan diferente de la sintaxis para degradados lineales que acabamos de estudiar. Usaremos la función radial-gradient() que incluye un parámetro para la forma cuyo valor puede ser circle o ellipse.



Código 3-17

Creación de un degradado radial.

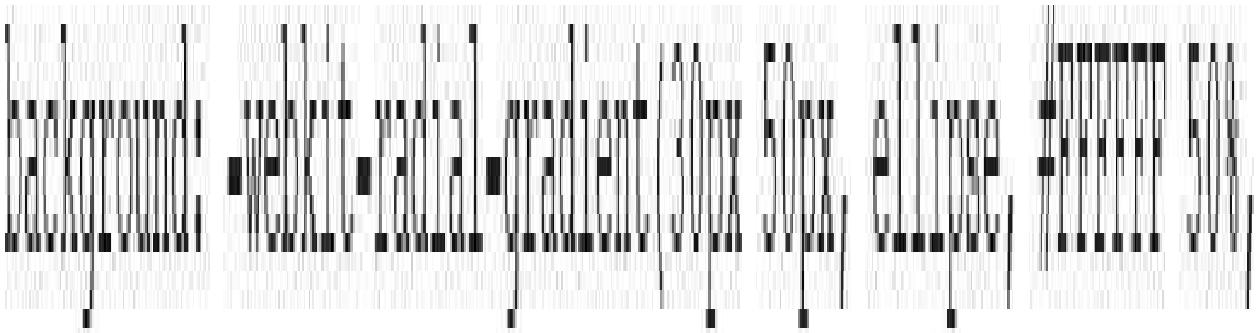
El primer valor en la función radial-gradient() del Código 3-17 determina la posición de inicio. Esta posición es el origen del degradado y se puede declarar ya sea como píxeles, como un porcentaje o una combinación de las palabras clave center, top, bottom, left y right.

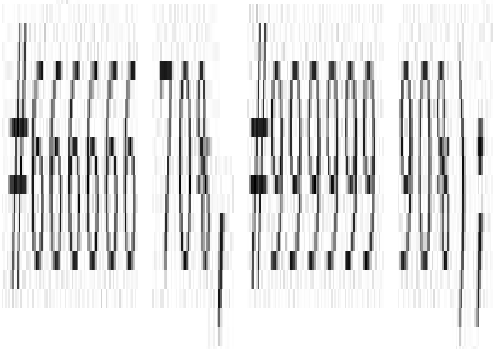
CSS Styles Web 2.0

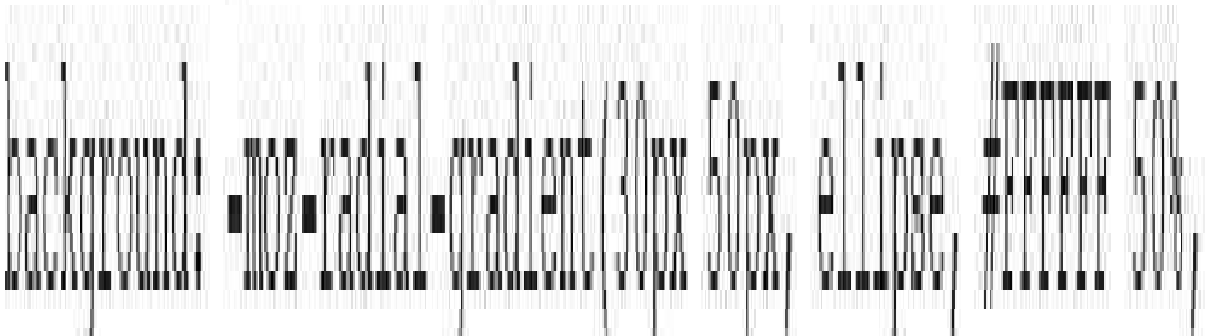
Figura 3-16

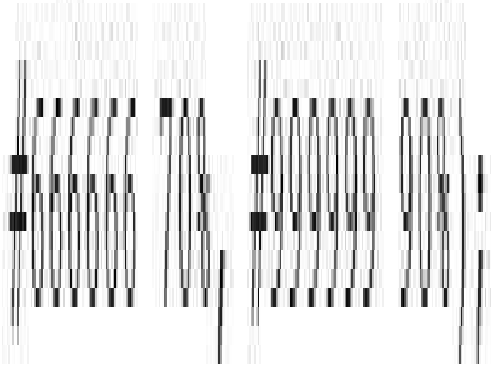
Degradado radial.

Excepto por el parámetro de forma (círculo o elipse), el resto de la función es exactamente igual que linear-gradient(). La posición puede ser personalizada y podemos utilizar varios colores con el segundo valor de color-stop para determinar el límite de cada uno de ellos.









Código 3-18

Creación de un degradado multicolor radial.

css Styles Web 2.0

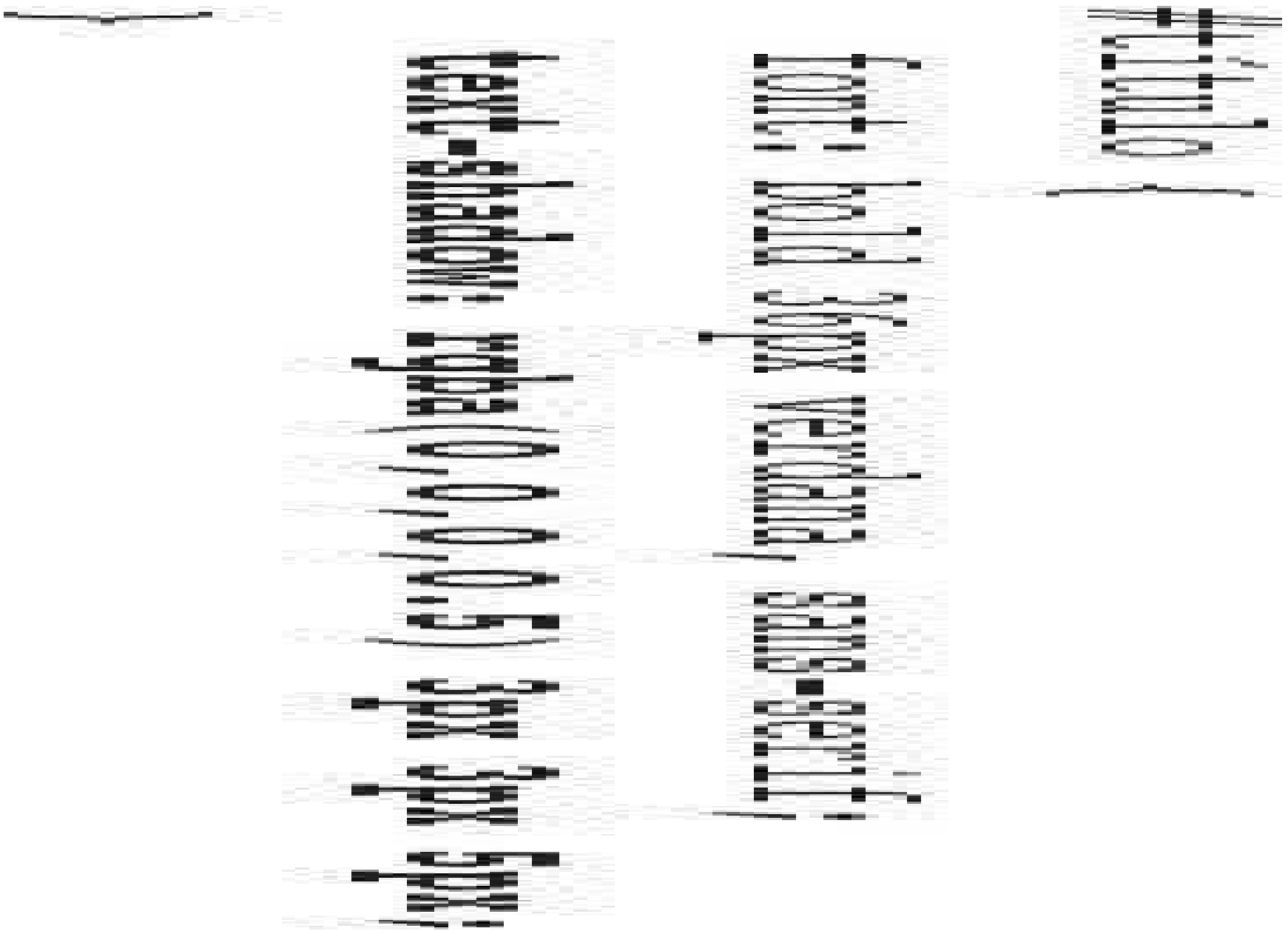
Figura 3-17

Uso de color-stop con un degradado radial.

3.1.9 rgb

Hasta este punto los colores han sido declarados como sólidos, con números hexadecimales o con la función rgb() para los decimales. CSS3 ha añadido una nueva función llamada rgba() que simplifica la declaración de colores y transparencias, y resuelve un problema anterior causado por la propiedad opacity.

El rgba() tiene cuatro atributos. Los tres primeros valores son similares a rgb() y simplemente declaran la combinación de colores. El último es para la opacidad y su valor puede estar comprendido entre 0 y 1, donde 0 es totalmente transparente y 1 totalmente opaco.



Código 3-19

Mejorar la sombra con transparencia.

El Código 3-19 contiene un ejemplo sencillo para demostrar cómo los efectos de sombra mejoran mediante el uso de la transparencia. Hemos sustituido la función rgb() por rgba() en la sombra del título de nuestro ejemplo del Código 3-9 para añadir un valor de opacidad de 0,5. Ahora la sombra del título se fusiona con el fondo de la caja creando un efecto más natural.

Hágalo usted mismo

Sustituya las reglas correspondientes del Código 3-9 con el contenido del Código 3-19 para probar el efecto en su navegador.

En las versiones anteriores de CSS, para hacer un elemento transparente era necesario utilizar diferentes técnicas para diferentes navegadores. Todas ellas presentaban el mismo problema: el valor de opacidad de un elemento era heredado por todos sus elementos hijos. Este problema fue resuelto con rgba() y ahora se puede asignar un valor de opacidad al fondo de una caja sin que se vea afectado su contenido.

3.1.10 hsla

Así como rgba() añade el valor de opacidad a la función rgb(), la función hsla() hace lo mismo para la anterior función hsl().

La función hsla() es simplemente otra función para aplicar el color a un elemento, pero es más intuitiva que rgba() y a algunos diseñadores les resultará más fácil crear un conjunto personal de colores utilizándola. La sintaxis es hsla (tono, saturación, luminosidad, opacidad).

```
#titulo {
    font: bold 36px verdana, sans-serif;
    text-shadow: rgba(0,0,0,0.5) 3px 3px 5px;
    color: hsla(120, 100%, 50%, 0.5);
}
```

Código 3-20

Creación de un color para el título con hsla().

El tono representa un color de una rueda imaginaria y se expresa en grados de 0 a 360. En torno a 0 y 360 están los rojos, cerca de 120 se encuentran los verdes y cerca de 240 están los azules. La saturación se representa como un porcentaje entre 0% (escala de grises) y 100% (color completamente saturado). La luminosidad también es un porcentaje, de 0% (totalmente oscuro) a 100% (completamente blanco), donde el valor 50% representa una luminosidad normal o media. Finalmente el último valor representa la opacidad tal como sucedía en rgba().

Hágalo usted mismo

Pruebe los diferentes valores disponibles para la propiedad align-content utilizando las reglas CSS del Código 2-55. Recuerde de nuevo agregar otras propiedades a los elementos, como fondo de color, altura o borde, para ver e identificar cada caja en la pantalla.

3.1.11 outline

La antigua propiedad outline se ha ampliado en CSS3 para incluir un desplazamiento, es decir, para crear un segundo borde separado del borde del elemento.

Hágalo usted mismo

Sustituya las líneas correspondientes del Código 3-2 con el código del Código 3-21 para probar el efecto en su navegador.

```
#contenedor {
   display: block;
   width: 500px;
   margin: 50px auto;
   padding: 15px;
   border: 1px solid #999999;
   background: #FFFFFF;

   outline: 2px dashed #000000;
   outline-offset: 15px;
}
```

Código 3-21

Añadir un borde externo para la caja de la cabecera.

En el Código 3-21 añadimos a los estilos asignados originalmente a la caja de la cabecera un borde exterior de 2 px con un desplazamiento de 15 px. La propiedad outline tiene características similares y utiliza los mismos parámetros que border. La propiedad outline-offset solo tiene un valor en píxeles.

CSS Styles Web 2.0

Figura 3-18

Adición de un segundo borde.

3.1.12 border-image

Los efectos logrados con las propiedades border y outline no son más que simples líneas y algunas opciones de configuración. La nueva propiedad border-image pretende superar estas limitaciones y permitir que sean los diseñadores quienes establezcan la calidad y la variedad de los bordes, ofreciendo la alternativa de utilizar imágenes personalizadas.

Hágalo usted mismo

Vamos a utilizar una imagen PNG de diamantes para poner a prueba esta propiedad. Siga el enlace para descargar el archivo diamond.png de nuestra página web (www.minkbooks.com/content/) y luego copie el archivo en la misma carpeta o directorio que el archivo CSS.

La propiedad border-image toma una imagen como patrón. De acuerdo con los valores proporcionados, la imagen es cortada como un pastel para obtener piezas y después estas piezas se colocan alrededor del objeto para construir el borde.

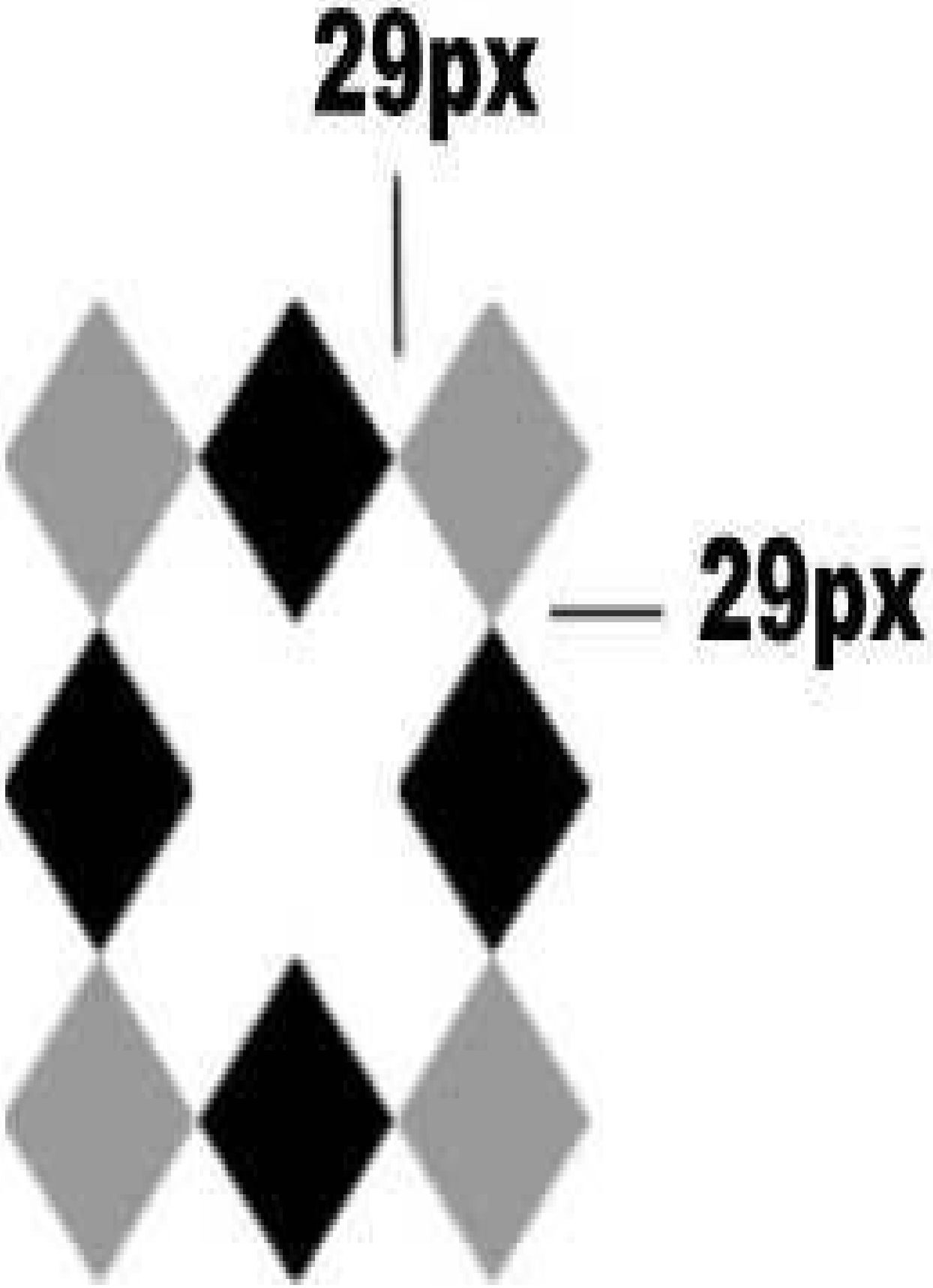

29px
29px

Figura 3-19

El patrón con el que construiremos el borde, con piezas de 29 px de ancho cada uno.

Para lograrlo tenemos que declarar tres atributos: el nombre del archivo de imagen y su ubicación, el tamaño de las piezas que se obtendrán del patrón y algunas palabras clave para especificar cómo se distribuyen las piezas alrededor del objeto.

```
#contenedor {
  display: block;
  width: 500px;
  margin: 50px auto;
  padding: 15px;
  border: 29px solid;
  border-image: url("diamonds.png") 29 stretch;
  }
```

Código 3-22

Creación de un borde personalizado para el cuadro de cabecera.

Con las modificaciones del Código 3-22 hemos establecido un límite de 29 px para el cajón superior y luego hemos cargado la imagen diamonds.png para construir la frontera. El valor 29 para la propiedad border-image declara el tamaño de las piezas, y stretch es uno de los métodos disponibles para distribuir estas piezas alrededor de la caja. Hay tres valores posibles para el último atributo. La palabra clave repeat establece que las piezas tomadas de la imagen se repitan tantas veces como sea necesario para cubrir el lado del elemento. En este caso, el tamaño de las piezas se conserva y la imagen se corta si no hay espacio suficiente para colocarla entera. La palabra clave round calcula la longitud del lado y luego estira las piezas para cubrir todo el lado y asegurarse al mismo tiempo de que ninguna de ellas se corta. Por último, la palabra clave stretch_(utilizado en el Código 3-22) se extiende en una sola pieza para cubrir todo el lado.

Hágalo usted mismo

Sustituya las líneas correspondientes del Código 3-2 con el Código 3-22 para probar el efecto en su navegador. Pruebe cada palabra clave para la propiedad border-image(repeat, round y stretch).

Aunque en el código anterior utilizamos la propiedad border para establecer el tamaño del borde, también se puede utilizar border-with para especificar un

tamaño diferente para cada lado del elemento. La propiedad border-with utiliza cuatro parámetros, con una sintaxis similar a las de margin y padding. Lo mismo sucede con el tamaño de cada pieza: pueden ser declarados hasta cuatro valores para obtener diferentes imágenes de diferentes tamaños a partir de un mismo patrón.

css Styles Web 2.0

Figura 3-20

Imagen de borde para la cabecera.

3.1.13 background

La propiedad background ha incorporado nuevos parámetros y características tales como el tamaño, la posición o el área de pintura, y puede incluir varias imágenes. La nueva sintaxis es background: color imagen posición tamaño repetir origen clip documento_adjunto, que es la forma abreviada de mencionar las siguientes propiedades:

background-color: Asigna un color de fondo a un elemento. El valor puede ser hexadecimal (por ejemplo, #FF0000), números decimales utilizando la función rgb(), por ejemplo, rgb (255,0,0), o una palabra clave como yellow, white, black, etc.

background-image: Esta propiedad asigna una o más imágenes para el fondo de un elemento. La dirección URL del archivo es declarada por la función url() (por ejemplo, url('bricks.jpg')). Si hay más de una imagen, los valores deben estar separados por una coma.

background-position: Declara la posición de partida de una imagen de fondo. Los valores se pueden especificar en píxeles o porcentajes utilizando una combinación de las siguientes palabras clave: center, left, right, top, bottom.

background-size: Declara el tamaño de la imagen de fondo. Los valores se pueden indicar como porcentaje, como píxeles o por las palabras clave cover y contain. Mientras cover escala la imagen hasta ajustarla al elemento en anchura o altura, contain escala la imagen para encajarla entera.

background-repeat: Determina la forma en la que la imagen de fondo se distribuye utilizando cuatro palabras clave: repeat, repeat-x, repeat-y y no-repeat. Mientras repeat repite la imagen horizontalmente y verticalmente, repeat-x y repeat-y lo hacen solo en el eje horizontal y el vertical respectivamente. Por último, no-repeat muestra la imagen de fondo solo una vez.

fondo-clip: Declara el área a pintar utilizando tres palabras clave: border-box, padding-box y content-box. La primera muestra la imagen hasta el borde de la caja, la segunda hasta relleno de la caja y la tercera solo hasta contenido de la caja.

background-origin: Utiliza las mismas palabras clave que background-clip para ajustar la imagen respecto al borde, el relleno o el contenido de la caja.

background-attachment: Esta propiedad determina si la imagen está fija o se desplaza con el resto de los elementos y para ello utiliza las palabras clave scroll y fixed. La primera es el valor por defecto y hace que la imagen se desplace con la página y la segunda, fixed, fija la imagen de fondo en el lugar en el que fue declarada.

```
#contenedor {
  display: block;
  width: 500px;
  margin: 50px auto;
  padding: 15px;
  border: 1px solid #999999;

  background: #cccccc url("ladrillos.jpg") 10px 10px / 510px 55px
 no-repeat scroll border-box;
}
```

Código 3-23

Generar de un fondo personalizado para el cajón superior.

En el Código 3-23 se ha creado un fondo usando la propiedad background. Observe que los valores están separados por un espacio, salvo position y size, que están separados por una barra. Se ha declarado el color, la imagen, la posición y el tamaño de la imagen de fondo en ese orden. Los valores no-repeat, scroll y border-box del final corresponden a los valores de background-repeat, background-attachment y background-origin.

Estilos CSS3 Web2.0

Figura 3-21

Prueba de la propiedad background.

Es posible intercambiar algunos valores o utilizar background declarando solo algunos de ellos, por ejemplo: background: #CCCC99.

Hágalo usted mismo

Sustituya las líneas correspondientes del Código 3-2 con el Código 3-23 para probar el efecto en su navegador. Tenga en cuenta que algunos de los nuevos parámetros para la propiedad background actualmente solo han sido implementados por Google Chrome. Para descargar la imagen utilizada como fondo en este ejemplo, vaya a www.minkbooks.com/content/.

3.1.14 Columnas

CSS3 incorpora un grupo de propiedades para facilitar la creación de columnas dentro de una caja. Gracias a ellas ahora podemos distribuir un contenido en varias columnas mediante una simple regla.

```
#contenedor {
  display: block;
  width: 500px;
  margin: 50px auto;
  padding: 15px;
  border: 1px solid #999999;
  background: #FFFFFF;

  column-count: 2;
  column-gap: 10px;
  column-rule: 1px solid #000000;
}
```

Código 3-24

Generación de columnas de texto.

La propiedad column-count, aplicada en el Código 3-24, declara el número de columnas a crear. Con column-gap declaramos la diferencia entre las columnas y, finalmente, column-rule especifica una división visible entre las columnas que aparecen en pantalla. En nuestro ejemplo es una línea de color negro sólido de 1 px de ancho. La sintaxis de esta última propiedad es la misma que para la propiedad border, es decir, width style color (separados por un espacio).

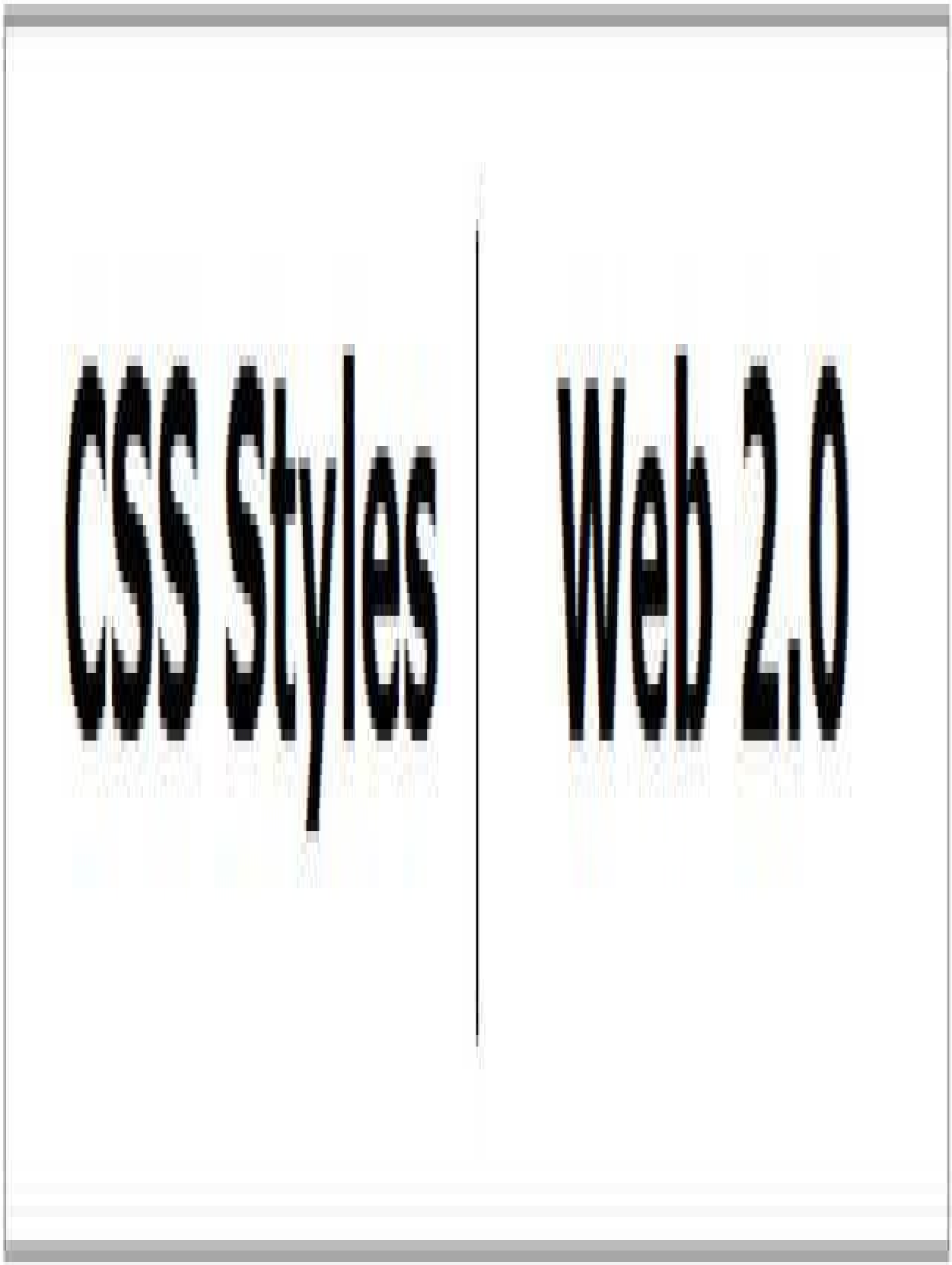

CSS Styles
Web 2.0

Figura 3-22

Cabecera de dos columnas.

Hay otras propiedades disponibles para personalizar las columnas. Algunas de las más interesantes son column-width y column-span:

column-width: Declara un ancho específico para las columnas. Los valores pueden ser auto (por defecto) o una longitud en cualquier unidad válida en CSS, por ejemplo píxeles.

column-span: Se aplica a los elementos dentro de la caja y determina si el elemento se colocará en el interior de una columna o será distribuido en varias columnas. Los valores posibles son all, es decir, “todas las columnas”, y none, aplicado por defecto.

Hágalo usted mismo

Pruebe este ejemplo utilizando las reglas CSS del Código 3-2 y verá el título de la cabecera dividido en dos columnas, como se muestra en la Figura 3-22. Recuerde añadir los prefijos correspondientes a las propiedades -colum- (por ejemplo, -webkit-, -moz- etc.).

3.1.15 Filtros

Los filtros permiten añadir efectos a una imagen. Han sido utilizado por los diseñadores gráficos durante mucho tiempo y aunque podemos encontrarlos en programas de edición de fotografía o editores de vídeo, por decir algunos ejemplos, los programadores no contaban con que estuvieran disponibles en la Web, salvo que fuera después de pasar horas programándolos. Sin embargo CSS3 cambió esta realidad al añadir varias funciones nuevas de la antigua propiedad filter para modificar no solo las imágenes, sino también cualquier otro elemento del documento.

```
#contenedor {
  display: block;
  width: 500px;
  margin: 50px auto;
  padding: 15px;
  border: 1px solid #999999;

  background: #CCCC99 url("bricks2.jpg");
  -webkit-filter: blur(5px);
}
```

Código 3-25

Aplicar un filtro a la cabecera.

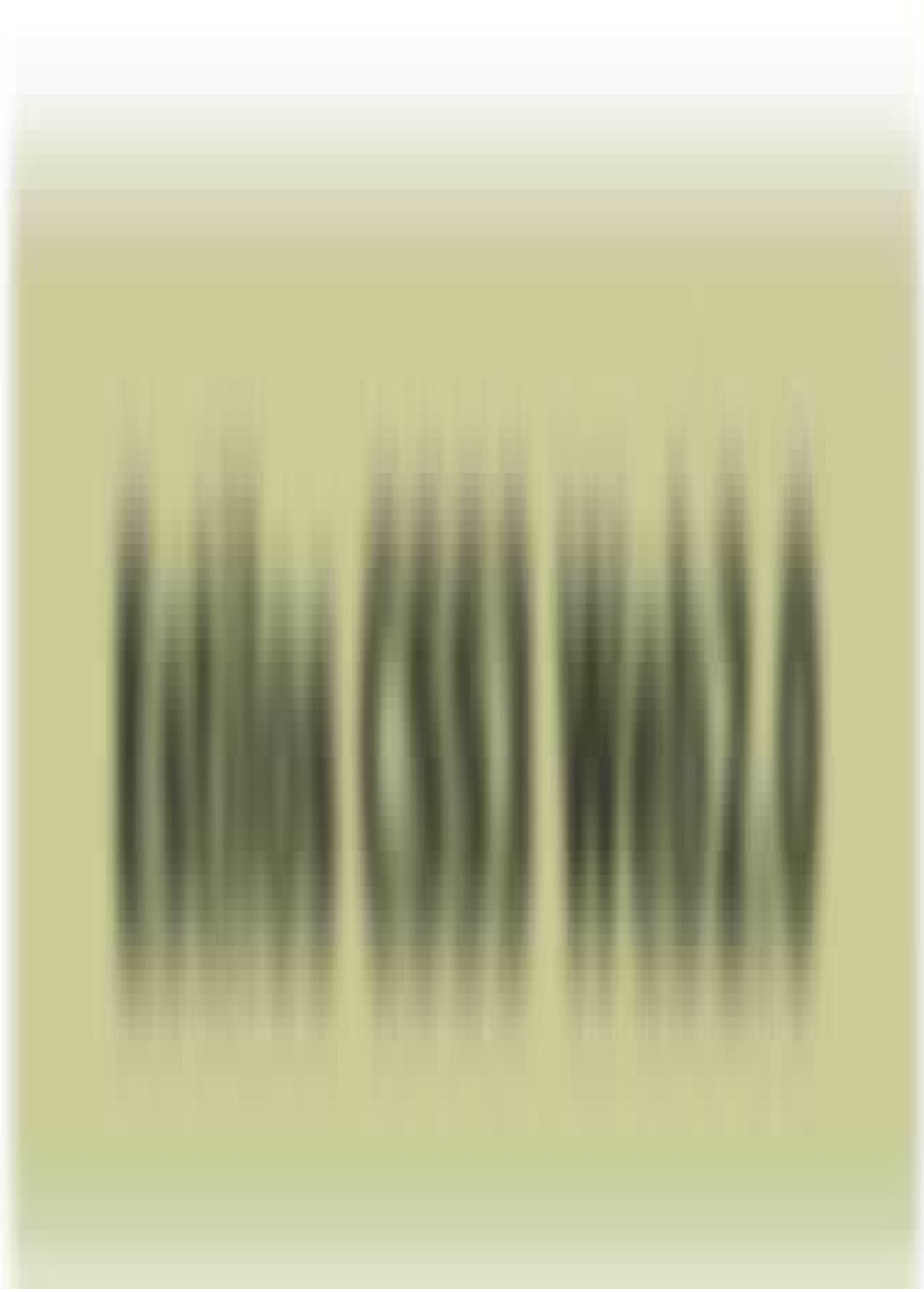


Figura 3-23

Cabecera con desenfoque.

La propiedad filter del Código 3-25 aplica el filtro blur() para proporcionar un efecto de desenfoque a la cabecera del documento (y a todo su contenido). Las funciones ahora disponibles para esta propiedad son:

blur(valor): Produce un efecto de desenfoque. Necesita un valor en píxeles de 1px a 10px.

grayscale(valor): Convierte gradualmente los colores del elemento a una escala de grises. Se necesita un número decimal de 0.1 a 1.

drop-shadow(x, y, tamaño, color): Aplica una sombra simple al elemento. Los atributos X y Y determinan la distancia entre la sombra y el elemento, el atributo tamaño especifica las dimensiones de la sombra y color declara su color.

sepia(valor): Proporciona a los colores del elemento un tono sepia. Necesita un número decimal de 0.1 a 1.

brightness(valor): Cambia la luminosidad del elemento. Necesita un número decimal de 0.1 a 10.

contrast(valor): Cambia el contraste del elemento. Necesita un número decimal de 0.1 a 10.

hue-rotate(valor): Aplica un giro a la tonalidad del elemento. Necesita un valor en grados de 1 a 360.

invert(valor): Invierte los colores del elemento y produce un negativo. Necesita un número decimal de 0.1 a 1.

saturarate(valor): Satura los colores del elemento. Necesita un número decimal de 0.1 a 1.

opacity(valor): Produce un efecto de opacidad. Necesita un número decimal de 0 to 1 (donde 0 establece transparencia completa y 1 opacidad completa).

Hágalo usted mismo

Sustituya el texto correspondiente al Código 3-2 con el Código 3-25 para probar el efecto en su navegador. Sustituya la función blur() con cualquier otra que desee aplicar.

Importante

De momento, los filtros solo están disponibles en Google Chrome. La aplicación se encuentra todavía en fase experimental, por lo que debe utilizar el prefijo -webkit- para aplicar la propiedad filter.

3.2 Transformar

Una vez creados, los elementos HTML se convertían en sólidos bloques casi inamovibles. Era posible moverlos utilizando códigos Javascript personalizados o algunas bibliotecas populares, como jQuery (www.jquery.com), pero hasta que CSS3 proporcionó las propiedades de transform y transition, no hubo un procedimiento estándar que facilitara el proceso. Ahora solo hace falta conocer algunos parámetros de estas propiedades para lograr que un sitio web sea tan dinámico como la imaginación de quien lo crea.

La propiedad transform puede aplicar cuatro transformaciones básicas a un objeto: scale, rotate, skew y translate. Veamos cómo funciona.

3.2.1 transform: scale

La función scale() usa dos parámetros, X para la escala horizontal y Y para la escala vertical. Si proporciona solo un valor, éste se aplica a ambos parámetros.

```
#contenedor {
   display: block;
   width: 500px;
   margin: 50px auto;
   padding: 15px;
   border: 1px solid #999999;
   background: #FFFFFF;

   -webkit-transform: scale(2);
   -moz-transform: scale(2);
}
```

Codigo 3-26

Escalar la caja de la cabecera.

En el ejemplo del Código 3-26, tomamos los estilos básicos que se utilizan en el Código 3-2 para el cuadro de cabecera y luego la ampliamos al doble de su tamaño. La propiedad acepta valores enteros y valores decimales, y se calcula utilizando una matriz. Los valores entre 0 y 1 reducen el elemento, 1 mantiene las proporciones originales y los valores mayores que 1 aumentan las dimensiones del objeto de forma lineal. Usando valores negativos se obtiene un efecto interesante:

Hágalo usted mismo

Sustituya las líneas correspondientes del Código 3-2 con el del Código 3-26 o el del Código 3-27 en su documento Javascript para probar el efecto en su navegador.

```
#contenedor {
   display: block;
   width: 500px;
   margin: 50px auto;
   padding: 15px;
   border: 1px solid #999999;
   background: #FFFFFF;

   -webkit-transform: scale(1,-1);
   -moz-transform: scale(1,-1);
}
```

Código 3-27

Creación de una imagen en espejo con escala.

En el Código 3-27, se declaran dos parámetros para modificar la escala del elemento contenedor. El primer valor, 1, mantiene la proporción original en la dimensión horizontal y el segundo valor mantiene la proporción original, pero invierte el elemento verticalmente para producir un efecto de espejo.

Figura 3-24

Imagen en espejo creada con scale().

Hay dos funciones similares a scale, pero restringidas a una sola dimensión: scaleX() y scaleY(). Estas funciones, desde luego, utilizan un solo parámetro.

3.2.2 transform: rotate

La función rotate() gira el elemento y su contenido en el sentido de las agujas del reloj. El valor debe ser especificado en grados usando la abreviatura del nombre de la unidad en inglés: deg. Por ejemplo: 30deg.

Hágalo usted mismo

Sustituya el texto correspondiente del Código 3-2 con el del Código 3-28 para probar el efecto en su navegador.

```
#contenedor {
  display: block;
  width: 500px;
  margin: 50px auto;
  padding: 15px;
  border: 1px solid #999999;
  background: #FFFFFF;

  -webkit-transform: rotate(30deg);
  -moz-transform: rotate(30deg);
}
```

Código 3-28

Girar la caja.

Si se especifica un valor negativo, cambiará la dirección en la se hace girar que el elemento.

3.2.3 transform: skew

La función de skew() cambia la simetría del elemento en ambas dimensiones utilizando también valores en grados (deg).

```
#mainbox {
   display: block;
   width: 500px;
   margin: 50px auto;
   padding: 15px;
   border: 1px solid #999999;
   background: #FFFFFF;

   -webkit-transform: skew(20deg);
   -moz-transform: skew(20deg);
}
```

Código 3-29

Sesgo horizontal.

Esta función tiene dos parámetros, pero a diferencia de otras funciones, para cada parámetro de skew() solo afecta a una dimensión, por lo tanto los parámetros son independientes uno de otro. En el Código 3-29, llevamos a cabo una operación de transform a la caja de cabecera con el fin de sesgar la misma. solo el primer parámetro se declara, por lo que solo la dimensión horizontal se modifica. Si utilizan ambos parámetros, se alteran considerablemente las dimensiones del objeto. Alternativamente, podríamos usar las funciones independientes para este fin: skewX() y skewY().

Hágalo usted mismo

Sustituya el texto correspondiente del Código 3-2 con el del Código 3-29 para probar el efecto en su navegador.

Estilos CSS3 Web2.0

Figura 3-25

Efecto skew().

3.2.4 transform: translate

Similar a las antiguas propiedades CSS top y left, la función translate( ) mueve el elemento a una nueva ubicación en la pantalla.

```
#contenedor {
  display: block;
  width: 500px;
  margin: 50px auto;
  padding: 15px;
  border: 1px solid #999999;
  background: #FFFFFF;
  -webkit-transform: translate(100px);
  -moz-transform: translate(100px);
}
```

Código 3-30

Desplazamiento de la caja contenedora hacia la derecha.

La función translate() considera la pantalla como una cuadrícula de píxeles donde la posición original del elemento se usa como punto de referencia. La esquina superior izquierda del elemento es la posición 0, 0, de modo que los valores negativos mueven el objeto a la izquierda o por encima de la posición original, y los valores positivos lo mueven a la derecha o hacia abajo.

Hágalo usted mismo

Sustituya el texto correspondiente del Código 3-2 con el del Código 3-30 para probar el efecto en su navegador.

El Código 3-30 se desplaza la cabecera 100 píxeles a la derecha desde su posición original. Pueden declararse dos valores para esta función con el objetivo de mover el elemento horizontal y verticalmente, pero también pueden usarse las funciones translateX() y translateY() de forma independiente.

3.2.5 Transformar todo en un elemento

A veces puede ser útil aplicar varias transformaciones a un solo elemento. Para obtener una propiedad transform compuesta, solo tiene que separar las funciones con un espacio:

```
#contenedor {
  display: block;
  width: 500px;
  margin: 50px auto;
  padding: 15px;
  border: 1px solid #999999;
  background: #FFFFFF;

  -webkit-transform: translateY(100px) rotate(45deg) scaleX(0.3);
  -moz-transform: translateY(100px) rotate(45deg) scaleX(0.3);
}
```

Código 3-31

Mover, escalar y rotar el elemento en una sola línea.

Tenga presente que el orden de las transformaciones es importante. Debido a que algunas funciones modifican el punto de origen y el centro del objeto, es posible que cambien los parámetros con los que operan el resto de las funciones.

Hágalo usted mismo

Sustituya el texto correspondiente del Código 3-2 con el del Código 3-31 para probar el efecto en su navegador.

3.2.6 Transformaciones dinámicas

Lo que hemos aprendido hasta ahora modifica nuestra página web, pero sus elementos permanecen estáticos. Sin embargo, podemos sacar ventaja de la combinación de transformaciones y pseudo-clases para convertir nuestro documento en una aplicación dinámica.

```
#contenedor {
   display: block;
   width: 500px;
   margin: 50px auto;
   padding: 15px;
   border: 1px solid #999999;
   background: #FFFFFF;
}
#contenedor:hover{
   -webkit-transform: rotate(5deg);
   -moz-transform: rotate(5deg);
}
```

Código 3-32

Respuesta a la actividad del usuario.

La norma original de la caja contenedora introducida en el Código 3-2 sigue siendo la misma, pero en el Código 3-32 se le añade una nueva regla para aplicar el efecto transform utilizando la pseudo-clase :hover. El resultado es que cada vez que el puntero del ratón está sobre la cabecera, transform la hace girar 5° y, cuando el puntero del ratón sale de la caja, la cabecera regresa a su posición anterior. De este modo se logra una animación básica pero útil utilizando solo propiedades CSS.

Hágalo usted mismo

Sustituya el texto correspondiente del Código 3-2 con el del Código 3-32 para probar el efecto en su navegador.

Conceptos básicos

La pseudo-clase :hover, así como el resto de pseudo-clases estudiadas en el Capítulo 2, añade un efecto especial. En este caso, los estilos se aplican solo cuando el puntero del ratón está sobre el elemento al que hace referencia la regla. Para obtener una lista completa de pseudo-clases visite nuestro sitio web y

siga los enlaces de este capítulo.

3.2.7 Transformaciones 3D

De la misma manera que se pueden hacer transformaciones de dos dimensiones, también se pueden realizar operaciones de tres dimensiones sobre los elementos HTML. CSS3 ofrece funciones y propiedades que producen sorprendentes efectos 3D para nuestras páginas web solo con unas pocas líneas de código.

```
#contenedor {
  display: block;
  width: 500px;
  margin: 50px auto;
  padding: 15px;
  border: 1px solid #999999;
  background: #FFFFFF;

  -webkit-transform: perspective(500px) rotate3d(0, 1, 0, 45deg);
  -moz-transform: perspective(500px) rotate3d(0, 1, 0, 45deg);
```

Código 3-33

Aplicación de un efecto tridimensional a la cabecera

Como se puede ver en el Código 3-33, hay nuevas funciones disponibles para la propiedad transform que producen transformaciones 3D. En este primer ejemplo se asigna perspectiva a la cabecera del documento que contiene el Código 3-1 y se produce una rotación 3D.

Estilos CSS3 Web2.0

Figura 3-26

Efecto 3D.

Exceptuando perspective(), el resto de las funciones 3D disponibles para la propiedad transform son similares a las de los efectos en 2D.

perspective(): Añade profundidad a la escena, haciendo los elementos más grandes cuando están más cerca del espectador. Para aplicar esta función primero es necesario ser capaz de ver los efectos 3D en la pantalla.

translate3d(x, y, z): Mueve el elemento a una nueva posición en el espacio 3D. Necesita tres valores en píxeles para los ejes X, Y y Z.

scale3d(x, y, z): Proporciona una nueva escala en el espacio 3D. Necesita tres valores en números decimales para establecer la escala de los ejes X, Y y Z. Como en las transformaciones 2D, un valor igual a 1 mantiene la escala original.

Rotar3D(x, y, z, ángulo): Hace girar el elemento de acuerdo con los valores proporcionados para cada eje y ángulo. Hay que indicar los valores de los ejes en números decimales y el ángulo en grados (por ejemplo, 30deg). Los valores para los ejes determinan un vector para la rotación, por lo que lo realmente importante es la relación entre los valores y no los valores en sí mismos. Por ejemplo, rotate3D(5, 2, 6, 30deg) producirá el mismo efecto que rotate3D(50, 20, 60, 30deg), debido a que el vector resultante es el mismo.

También en este caso podemos aplicar todas las transformaciones a la vez.:

```
#contenedor {
  display: block;
  width: 500px;
  margin: 50px auto;
  padding: 15px;
  border: 1px solid #999999;
  background: #FFFFFF;
  -webkit-transform: perspective(500px) translate3d(0, 0, -300px)
                     rotate3d(0, 1, 0, 50deg);
  -moz-transform: perspective(500px) translate3d(0, 0, -300px)
                  rotate3d(0, 1, 0, 50deg);
```

Código 3-34

Uso de translate y rotate3D.

Hay otras propiedades disponibles que permiten conseguir un efecto más realista:

perspective: Es igual que la función la perspective() pero funciona en una caja padre. Crea una caja contenedora y aplica el efecto de perspectiva a sus elementos hijos.

perspective-origin: Cambia las coordenadas X y Y del espectador. Necesita dos valores en píxeles o porcentaje, o las palabras clave center, left, right, top y bottom. Los valores por defecto son 50% 50%.

backface-visibility: Determina si la parte trasera de un elemento será visible o no. Tiene dos valores: visible o hidden, y visible es el valor por defecto.

```
body {
  text-align: center;

  -webkit-perspective: 500px;
  -moz-perspective: 500px;

  -webkit-perspective-origin: 50% 90%;
  -moz-perspective-origin: 50% 90%;
}

#contenedor {
  display: block;
  width: 500px;
  margin: 50px auto;
  padding: 15px;
  border: 1px solid #999999;
  background: #FFFFFF;

  -webkit-transform: rotate3d(0, 1, 0, 135deg);
  -moz-transform: rotate3d(0, 1, 0, 135deg);
}

#título {
  font: bold 36px verdana, sans-serif;
}
```

Código 3-35

Declaración de un inicio diferente para el espectador.

La propiedad perspective tiene que ser asignada a la caja contenedora del elemento que desea modificar. En el Código 3-1, la el elemento padre de la cabecera es el cuerpo del documento y a éste le hemos aplicado la propiedad perspective(). Hubiéramos obtenido el mismo resultado aplicando la función directamente al elemento hijo, la cabecera, pero al hacerlo de esta manera, hemos sido capaces de utilizar la propiedad perspective-origin y cambiar así también las coordenadas para el espectador.

Hágalo usted mismo

Sustituya el Código 3-2 de su documento Javascript con el del Código 3-35 para probar el efecto en su navegador. Trate de añadir la característica backface-visibility con el valor hidden al elemento contenedor para hacer que la cabecera sea invisible cuando esté al revés.

Estilos css3 web2.0

Figura 3-27

Efecto 3D mediante la propiedad perspective.

3.3 Transiciones

Ya hemos visto que con CSS3 es fácil aplicar efectos llamativos con transformaciones dinámicas. Sin embargo, una animación real requiere una transición entre las dos etapas del proceso.

Importante

En el Código 3-36 llevamos a cabo una transición con la propie dad transform. En el momento de preparar este libro, no todas las propiedades CSS son soportadas por la propiedad transition ni por todos los navegadores, pero esto va a cambiar. Téngalo en cuenta y realice sus propias pruebas o visite el sitio web de los navegadores para encontrar más información.

La propiedad transition fue creada para suavizar el cambio y, así, mágicamente, crear el resto de los pasos implícitos en el movimiento. Al añadir esta propiedad, forzamos al navegador a manejar la situación, recrear esos pasos invisibles y generar una suave transición de un estado a otro.

```
#contenedor {
  display: block;
  width: 500px;
  margin: 50px auto;
  padding: 15px;
  border: 1px solid #999999;
  background: #FFFFFF;

  -webkit-transition: -webkit-transform 1s ease-in-out 0.5s;
  -moz-transition: -moz-transform 1s ease-in-out 0.5s;
}
#contenedor:hover {
  -webkit-transform: perspective(500px) rotate3d(0, 1, 0, 360deg);
  -moz-transform: perspective(500px) rotate3d(0, 1, 0, 360deg);
}
```

Código 3-36

Creación de una rotación en 3D con transición 3D.

Como se puede ver en el Código 3-36, la propiedad transition puede tomar hasta cuatro parámetros separados por espacios. El primer valor es la propiedad que se usará para crear la transición (transform en nuestro ejemplo). Es necesario porque aunque varias propiedades pueden cambiar al mismo tiempo, probablemente solo será necesario crear pasos para una de ellas. El segundo parámetro establece el tiempo en el que la transición ejecutará, desde la posición inicial hasta la posición final (un segundo en nuestro ejemplo). El tercer parámetro puede ser alguna de estas cinco palabras clave: ease, linear, ease-in, ease-out o ease-in-out. Estas palabras clave determinan cómo se llevará a cabo la transición, sobre la base de una curva de Bézier. Cada palabra clave representa una curva de Bézier diferente y la única manera de saber cuál es la mejor para una transición es realizando una prueba en la pantalla.

El último parámetro de la propiedad de transition es el retardo. Indica cuánto tiempo se tardará la transición para comenzar.

También es posible utilizar la palabra clave all para generar una transición de todas las propiedades que cambian en un elemento o declarar varias propiedades separándolas por comas.

Hágalo usted mismo

Sustituya el texto correspondiente en el Código 3-2 con el Código 3-36 para probar el efecto en su navegador.

Importante

Actualmente los fabricantes de navegadores están trabajando en la eliminación de los prefijos para las propiedades transform, transition y animation, pero es probable que aún tenga que añadir los prefijos para estos ejemplos. La función @keyframes utiliza el prefijo entre el símbolo @ y el nombre (por ejemplo, @-webkit-keyframes).

3.4 Animaciones

Probablemente las animaciones constituyen el efecto más complejo que se puede lograr con CSS. En la sección anterior vimos cómo crear una animación básica mediante la propiedad transition, pero el proceso implicaba solo dos pasos: el estado inicial y el estado final.

Una verdadera animación exige la declaración de varias tramas, como en una película y CSS3 permite hacerlo con la función @keyframes y la propiedad animation. La función define la animación y la propiedad proporciona los parámetros de configuración, como se muestra en el Código 3-37.

```
body {
  text-align: center;
}

#contenedor {
  display: block;
  width: 500px;
  margin: 50px auto;
  padding: 15px;
  border: 1px solid #999999;
  animation: myanimation 1s ease-in-out 0s infinite normal none;
}

@keyframes myanimation {
  0% {
    background: #FFFFFF;
  }
  50% {
    background: #FF0000;
  }
  100% {
    background: #FFFFFF;
  }
}

#título {
  font: bold 36px verdana, sans-serif;
}
```

Código 3-37

Acercamiento a la animación de CSS.

Los fotogramas son definidos por la función @keyframes usando valores porcentuales y agrupando los estilos para cada fotograma con llaves. El valor 0% define el primer cuadro o el inicio de la película mientras el 100% define el último cuadro o el final estos valores pueden ser ignorados o también sustituidos con las palabras clave from (0%) y to (100%).

Importante

Recuerde que para probar estos ejemplos debe añadir a la propiedad de animation y a los fotogramas clave, es decir, los @keyframes, los prefijos que correspondan según su navegador. Deberá usar, por ejemplo: -webkit-animation, @-webkit-key-frames).

La animación puede comenzar en cualquier momento y pueden declararse todos los cuadros que se quiera.

```
body {
  text-align: center;
}

#contenedor {
  display: block;
  width: 500px;
  margin: 50px auto;
  padding: 15px;
  border: 1px solid #999999;

  animation: myanimation 1s ease-in-out 0s infinite normal none;
}

@keyframes myanimation {
  20% {
    background: #FFFFFF;
  }
  35% {
    transform: scale(0.5);
    background: #FFFF00;
  }
  50% {
    transform: scale(1.5);
    background: #FF0000;
  }
  65% {
    transform: scale(0.5);
    background: #FFFF00;
  }
  80% {
    background: #FFFFFF;

  }
}
#título {
  font: bold 36px verdana, sans-serif;
}
```

Código 3-38

Declarar más fotogramas para nuestra animación.

En el Código 3-38, la animación comienza en 20% y termina a 80%, con un total de cinco fotogramas.

Las propiedades de cada fotograma indican el estilo del elemento en cada paso del proceso de animación pero no cómo se desarrollará la animación. Para configurarla está la propiedad animation. Esta propiedad permite aplicar varias propiedades disponibles para el controlar el proceso de animación:

animation-name: Proporciona el nombre que se utiliza para crear los fotogramas en la función @keyframe y conectar así de forma efectiva el fotograma con la configuración de la animación. Se puede utilizar para configurar varias animaciones a la vez declarando los nombres separados por comas.

animation-duration: Se utiliza para declarar cuánto tiempo durará cada ciclo de la animación. El valor debe ser especificado en segundos (por ejemplo, 1s para unt segundo).

animation-timing-function: Funciona exactamente igual que en la propiedad de transición que estudiamos antes. Determina cómo se llevará a cabo el proceso de animación a partir de una curva de Bézier declarada por las palabras clave ease, linear, ease-in, ease-out y ease-in-out.

animation-delay: Retrasa el inicio de la animación. Debe ser declarada en segundos (por ejemplo, 1s para un segundo), con un valor por defecto de 0.

animation-iteration-count: Declara el número de veces que se ejecuta la animación. Necesita un número entero para el valor o la palabra clave infinite, que ejecuta la animación indefinidamente. El valor por defecto es 1.

animation-direction: Establece la dirección de la animación. Puede tomar cuatro valores: normal, reverse, alternate y alternate-reverse. Por defecto se aplica el valor normal, que no produce ningún cambio. El segundo valor, reverse, invierte la dirección de la animación, mostrando los fotogramas en la dirección opuesta hayan sido declaradas. El valor alternate mezcla los ciclos de animación, reproduciendo los que tienen un índice impar en dirección normal y el resto en dirección opuesta. Por último, el valor alternate-reverse simplemente hace lo mismo que alternate, pero al revés.

animación-fill-mode: Esta propiedad puede tomar cuatro valores para definir la forma en la que la animación va a interactuar con los estilos del elemento: none, forwards, backwards y both. El valor none se aplica por defecto y no afecta a los estilos del elemento. El valor forwards mantiene el estilo del elemento con las propiedades aplicadas en el último fotograma de la animación (que se define con el valor 100% o la palabra clave to), mientras que backwards aplica el estilo del primer cuadro (definido por el valor 0% o la palabra clave from) tan pronto como se define la animación, es decir, antes de que se ejecute. Finalmente, el valor both produce ambos efectos.

Como hemos visto en nuestros ejemplos, es posible declarar todas estas propiedades a la vez utilizando la propiedad animation y la siguiente sintaxis): animation: name duration timing-function delay iteration-count direction fill-

mode. Recuerde que el orden es importante.

4 Javascript

4.1 Breve introducción a Javascript

Podemos comparar HTML5 con un edificio sostenido por tres columnas: HTML, CSS y Javascript. Ya hemos analizado los elementos incorporados al lenguaje HTML y las nuevas propiedades que convierten a CSS en la herramienta ideal para diseñadores. Ahora es el momento de dar a conocer lo que puede ser considerado como uno de los aspectos más potentes de esta especificación: Javascript.

Importante

Nuestro enfoque de Javascript en este libro es introductorio. Trabajaremos con algunas funciones básicas y otras complejas, pero solo aplicaremos los conceptos elementales necesarios para aprovechar las innovaciones introducidas en HTML5. Para ampliar sus conocimientos sobre el lenguaje, visite nuestro sitio web y siga los enlaces de este capítulo. Si está familiarizado con esta información, por favor siéntase con la libertad de saltarse los temas que ya conoce.

Javascript es un lenguaje interpretado utilizado para múltiples propósitos, y hasta hace poco fue considerado como un complemento de HTML y CSS. Una de las innovaciones que ayudaron a cambiar la percepción de Javascript fueron los nuevos motores de navegación desarrollados para acelerar el procesamiento de códigos. Estos motores convirtieron el código Javascript en código de máquina con el fin de alcanzar velocidades de ejecución similares a las aplicaciones de escritorio. Esta capacidad mejorada venció las anteriores limitaciones de rendimiento de Javascript y lo reafirmó como la mejor opción de codificación

para la Web.

Javascript necesita para funcionar un entorno de hospedaje, pero gracias a nuevos intérpretes y compiladores como V8 de Google, se ha convertido en un lenguaje más independiente y poderoso. Ahora se ejecuta casi en todas partes y es responsable de la revolución de Internet y del crecimiento del mercado móvil.

Para aprovechar las ventajas de esta prometedora arquitectura, se ha mejorado la portabilidad y la integración de Javascript. Además, en todos los navegadores se han incorporado por defecto completas interfaces de programación de aplicaciones (API) para asistir al lenguaje en funciones elementales. Estas nuevas API (como Web Storedad y Canvas, entre otras) son interfaces para librerías integradas en los navegadores. Hacen de este modo que características de gran alcance estén disponibles en cualquier lugar a través de sencillas y estandarizadas técnicas de programación, ampliando el alcance del lenguaje y facilitando la creación de software útil y atractivo para la Web.

Javascript es actualmente el lenguaje más exitoso y prometedor, y HTML5 lo está convirtiendo en una herramienta esencial para los desarrolladores web y de aplicaciones para móviles. En este capítulo estudiaremos los conceptos básicos de Javascript y incorporaremos códigos Javascrit en nuestros documentos HTML, al tiempo que presentaremos las novedades del lenguaje de forma que esté preparado para afrontar el resto del libro.

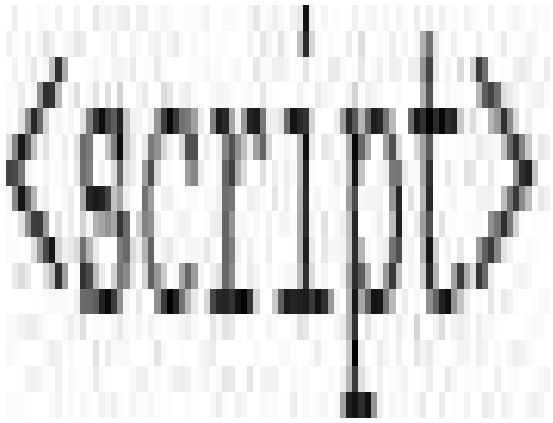
4.1.1 El lenguaje

Javascript es un lenguaje de programación, algo completamente diferente a HTML y CSS. HTML es un lenguaje de marcado, un código críptico que los navegadores interpretan para organizar la información y CSS se puede considerar como una hoja de estilos (aunque la nueva especificación lo ha convertido en una herramienta más dinámica). Javascript, por otra parte, es un lenguaje de código comparable a cualquier lenguaje de programación como C++ o Java. La única diferencia significativa entre la mayoría de los lenguajes de programación y Javascript es su naturaleza. Mientras que otros lenguajes modernos están orientados a objetos, Javascript es un lenguaje de código basado en prototipos, lo que, paradójicamente, lo hace más centrado en objetos que cualquier otro lenguaje.

Javascript puede realizar numerosas tareas, desde proporcionar instrucciones hasta calcular algoritmos complejos, pero la característica más importante, como en cualquier lenguaje de programación, es su capacidad de almacenar y procesar la información, cosa que realiza a través de variables.

4.1.2 Variables

La memoria de un ordenador o de dispositivo móvil es como un panal enorme con millones y millones de celdas disponibles para almacenar información. Esas celdas tienen una dirección, un número consecutivo que identifica a cada una de ellas. Tienen un espacio limitado y por lo general, para almacenar grandes cantidades de datos es necesaria la combinación de varias celdas. Debido a lo compleja que es la manipulación de este espacio de almacenamiento, los lenguajes de programación incorporan el concepto de variables para facilitar la identificación de cada valor almacenado en la memoria. Las variables son nombres asignados a una celda o grupo de celdas en la que van a ser almacenados los datos. Por ejemplo, si queremos almacenar el valor 5 en la memoria, tenemos que saber dónde se almacena ese número para poder recuperarlo más tarde. Si creamos una variable podremos identificar ese espacio por un nombre y recuperar el contenido de esa celda utilizando el nombre correspondiente. Para declarar una variable, Javascript utiliza la palabra clave var:


<script>

var minumero = 2;

</script>

Código 4-1

Declarar una variable de Javascript.

Conceptos básicos

La etiqueta <script> se utiliza para informar al navegador de que el código fuente que se encuentra entre la etiqueta de inicio y la de cierre fue escrito en Javascript. Es una de las formas posibles para incorporar código Javascript en un documento HTML, pero sepa que vamos a estudiar otras opciones más adelante.

En el Código 4-1, creamos la variable minumero y almacenamos en esa variable el valor 2. Este proceso se suele denominar asignar y efectivamente lo que hemos hecho es “asignar el valor 2 a la variable minumero”.

Javascript reserva un espacio en la memoria, almacena el número 2, crea una referencia a ese espacio y le asigna a esta referencia el nombre minumero. Así que cada vez que se usa la referencia minumero, se obtiene el número 2.

```
<script>

var minumero = 2;

alert(minumero);

</script>
```

Código 4-2

Usar el contenido de una variable.

En el Código 4-2 se crea la variable y luego se le asigna el valor 2. Ahora el contenido de la variable minumero es 2. Además, se muestra el contenido de la variable en pantalla usando el método alert().

Conceptos básicos

alert() es un método predefinido de Javascript que genera una ventana emergente para mostrar determinada información en la pantalla. El método muestra en la ventana cualquier valor declarado entre paréntesis (por ejemplo, alert(“Hola a todos”);). Estudiaremos éste y otros métodos similares más adelante.

Hágalo usted mismo

Para propósitos de prueba, no es necesario crear una estructura HTML completa. El contenido del Código 4-2 es más que suficiente para que los navegadores interpreten Javascript. Copie este código fuente en un archivo de texto vacío, guárdelo con un nombre y la extensión .html (por ejemplo, codigojs.html) y abra el archivo en su navegador. Verá una ventana emergente con el valor 2.

Si las variables se llaman “variables”, es precisamente porque no son constantes. Podemos cambiar el valor asignado a ellas en cualquier momento y ésta es, de hecho, su característica más importante.

```
<script>
    var minumero = 2;
    minumero = 3;
    alert(minumero);
</script>
```

Código 4-3

Asignación de un nuevo valor para la variable.

En el Código 4-3 se declara la variable y después se le asigna un nuevo valor. Finalmente el método alert() muestra el número 3. No es necesario usar la palabra clave var; para asignar el nuevo valor basta con el nombre de la variable seguido del signo =. En una situación real probablemente se utilizaría el valor de la variable para realizar una operación y luego se asignaría el resultado a la misma variable:

```
<script>
    var minumero = 2;

    minumero = minumero + 1;
    alert(minumero);
</script>
```

Código 4-4

Utilizando el valor almacenado en la variable.

En este ejemplo se suma 1 al valor actual de minumero y se asigna el resultado a la misma variable. Es lo mismo que sumar 2 + 1, con la diferencia de que al usar una variable en lugar de un número su valor puede cambiar continuamente, pues durante la ejecución del código el valor de las variables puede ser modificado. El número almacenado en la variable podría haber sido 5 en lugar de 2 gracias a un procedimiento anterior, y en ese caso el resultado de la suma habría sido 6 en lugar de 3. Esto nos da una idea del poder de las variables. Podemos realizar cualquier operación matemática o concatenar texto, y las variables siempre conservarán el último valor que se les haya asignado.

Conceptos básicos

Además del operador +, hay numerosos operadores disponibles en Javascript. Los más comunes son + (suma), - (resta), * (multiplicación) y / (división), pero el lenguaje también proporciona operadores lógicos, como && (y) y || (o), operadores de comparación como por ejemplo == (igual), != (diferente), < (menor que), > (mayor que), entre otros. Vamos a utilizar y estudiar varios de ellos a lo largo del libro. Para obtener una lista completa, por favor visite nuestro sitio web y siga los enlaces de este capítulo.

Las variables pueden almacenar números, texto y valores predefinidos, como valores booleanos (true –verdadero– o false –falso–), o valores elementales

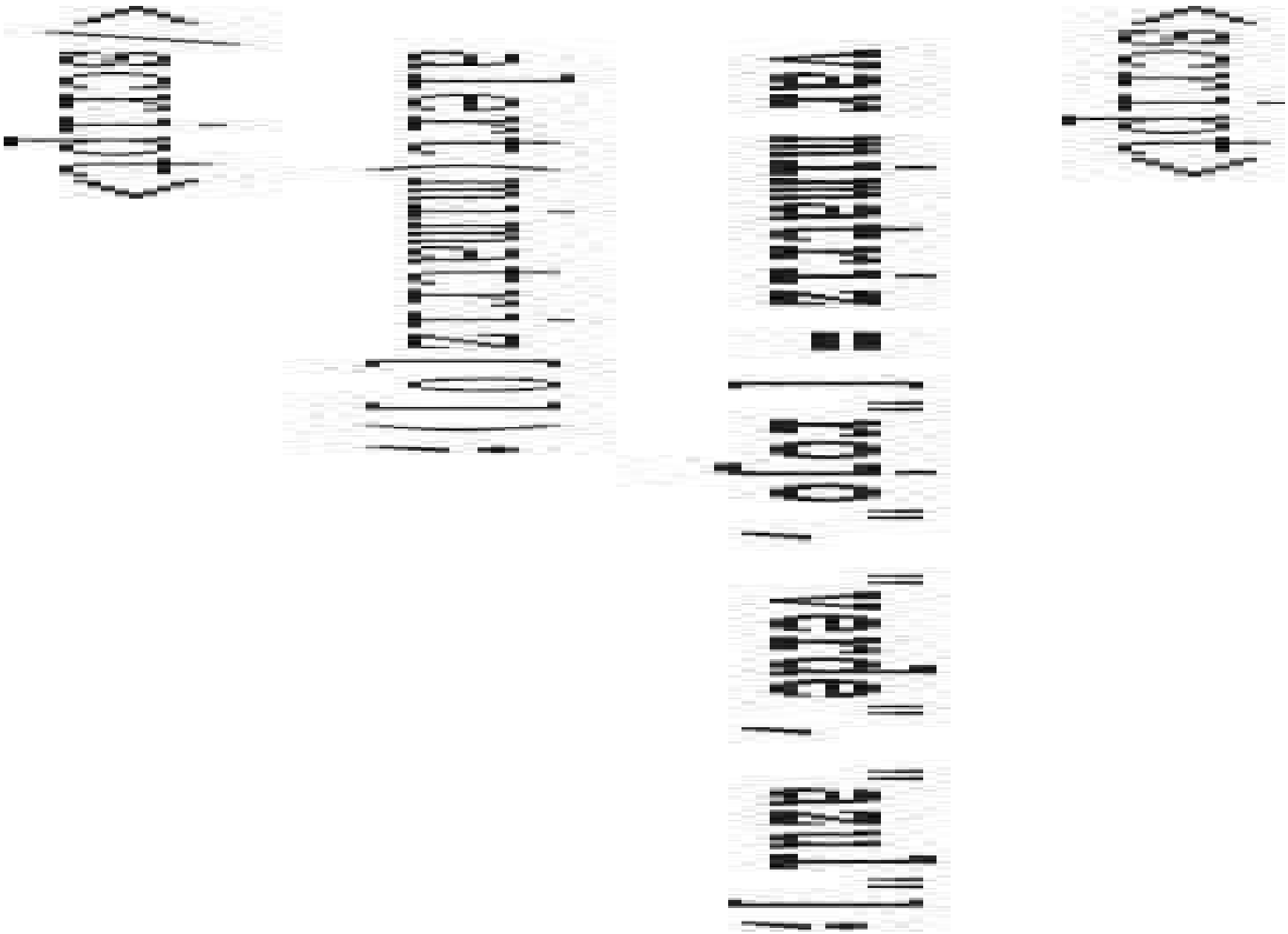
como null –nulo– o undefined –indefinido–. Los números se expresan igual que en los ejemplos anteriores, pero el texto debe ir entre comillas simples o dobles.

```
<script>

var mitexto = "¡Hola a todos!";

alert(mitexto);

</script>
```

Código 4-5

Asignación de texto a una variable.

Las variables también pueden almacenar varios valores a la vez en una estructura llamada matriz. Las matrices son variables multidimensionales que se crean utilizando una sencilla notación de corchetes separando cada valor con comas. Los valores se identifican más tarde utilizando un índice a partir de 0.

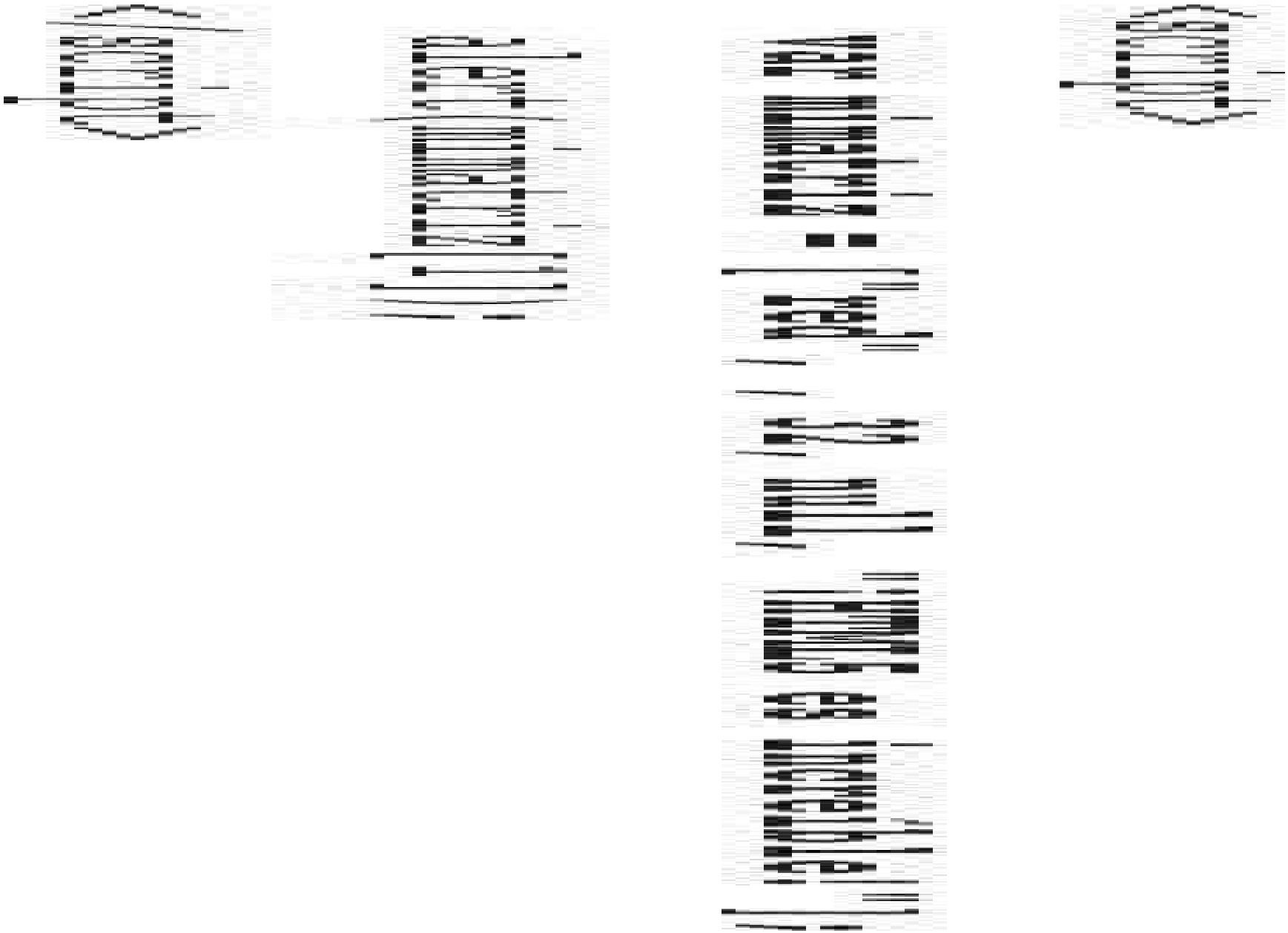


Código 4-6

Creación de variables multidimensionales.

En el Código 4.6 se crea una matriz llamada mimatriz con tres valores: rojo, verde y azul. Javascript asigna automáticamente el índice 0 al primer valor, 1 al segundo y 2 al tercer valor. Para utilizar la matriz y obtener esta información hay que mencionar el número de índice entre corchetes después del nombre de la variable. Por ejemplo, para obtener el primer valor de mimatriz tenemos que escribir mimatriz[0], como hicimos en el ejemplo anterior.

Las matrices, igual que las variables, pueden tomar cualquier tipo de valor. Podemos crear una matriz como la que en el Código 4-6 utilizando números, textos, valores booleanos o incluso declarar el valor como null o undefined.



Código 4-7

Uso de diferentes tipos de valores.

Hágalo usted mismo

Copia el Código 4-7 en un archivo de texto vacío, guárdelo con un nombre y la extensión .html (por ejemplo, Códigojs.html) y abra el archivo en su navegador. Cambie el índice para ver todos los valores de la matriz.

Conceptos básicos

null y undefined son valores nativos de Javascript. Básicamente, ambos representan la ausencia de valor, pero también se devuelve undefined cuando la propiedad de un objeto o un elemento de matriz no ha sido definida. En el ejemplo del Código 4-7, el segundo elemento de la matriz no ha sido definido. Si tratamos de recuperar este elemento, se devuelve el valor undefined.

Por supuesto, también puede realizar operaciones sobre los valores de una matriz y almacenar el resultado, como lo hicimos antes con las variables simples.

```
<script>
    var mimatriz = ["rojo", 32];
    alert(mimatriz[0]);
    mimatriz[0] = "color " + mimatriz[0];
    mimatriz[1] = mimatriz[1] + 10;
    alert(mimatriz[1] + " " + mimatriz[0]);
</script>
```

Código 4-8

Usar matrices.

Con el script del Código 4-8 se demuestra no solo cómo modificar el valor de una matriz, sino también cómo concatenar texto. Las matrices funcionan exactamente igual que las variables individuales, con la diferencia de que hay que mencionar el índice cada vez que desee utilizarlas. La declaración mimatriz[1] = mimatriz[1] + 10 le indica al intérprete de Javascript que tome el valor actual de mimatriz para el índice 1 (que es 32), sume 10 a ese valor y almacene el resultado (42) en la misma matriz y en el mismo índice. De este modo el valor de mimatriz[1] ahora es 42. El operador + funciona para números y textos, pero en el caso de los últimos concatena el texto y genera una nueva cadena de texto que incluye ambos valores. En nuestro ejemplo, el valor de mimatriz[0] era rojo pero más tarde se le añadió el texto color al comienzo de la cadena; por tanto ahora el valor de mimatriz en el índice 0 es color rojo.

Conceptos básicos

Al final de la última secuencia de comandos, el método alert() muestra ambos valores de la matriz separados por un espacio. Los valores y el espacio están unidos por el operador +. Este operador puede ser utilizado no solo para realizar operaciones sino también para concatenar valores y cadenas con fines de estilo (los números se convierten en cadenas de forma automática).

Javascript proporciona una forma sencilla de trabajar con matrices. Podemos

extraer o agregar valores mediante los siguientes métodos:

push(valor): Este método agrega un nuevo valor al final de la matriz. El atributo valor indica el valor que se añade.

shift(): Este método elimina y devuelve el primer valor de la matriz.

pop(): Este método elimina y devuelve el último valor de la matriz.

```
<script>
    var mimatriz = ["rojo", 32];
    mimatriz.push('coche');
    alert(mimatriz[2]);
    alert(mimatriz.shift());
</script>
```

Código 4-9

Adición y extracción de valores de la matriz.

El valor agregado por el método push() es añadido al siguiente índice de la matriz. En el Código 4-9, el valor añadido por el método push() adquiere el índice 2. Luego se utiliza precisamente este índice para mostrar el valor en pantalla. Al final de la secuencia de comandos, se extrae y muestra el primer valor de la matriz en pantalla. Tenga en cuenta que los valores extraídos con el método shift() dejan de formar parte de la matriz. Después de ejecutar la secuencia de comandos de este ejemplo, la matriz solo tiene los dos primeros valores de la izquierda: 32 y coche.

4.1.3 Condicionales y bucles

Un programa de ordenador no es tal si no permite establecer condiciones y ciclos complejos de procesamiento. Javascript ofrece un total de cuatro declaraciones para procesar código de acuerdo con condiciones determinadas por el programador: if, switch, for y while

```
<script>
  var mivariable = 9;
  if(mivariable < 10) {
    alert("El número es menor que 10");
  }
</script>
```

Código 4-10

Verificación de un estado con if.

Conceptos básicos

Se recomienda cerrar todas las líneas de código en Javascript con un punto y coma, aunque esto no es necesario después de una declaración de bloque (un bloque de código entre llaves).

La declaración simplemente comprueba la expresión entre paréntesis y procesa las instrucciones entre llaves si la condición es verdadera. En el script del Código 4-10, se asigna el valor 9 a mivariable; a continuación, utilizando la palabra if se compara la variable con el número 10. Si el valor de la variable es menor que 10, entonces el método alert() muestra un mensaje en la pantalla.

Podemos construir una declaración que nos permita realizar alguna acción, tanto si la condición es verdadera como si no lo es. La construcción if else comprueba si la condición entre paréntesis es cierta y realiza una acción diferente en cada caso.

```
<script>
  var mivariable = 9;
  if(mivariable < 10) {
    alert("El número es menor que 10");
  }else{
    alert("El número es igual o mayor que 10");
  }
</script>
```

Código 4-11

Verificación de las dos condiciones con if else.

En el ejemplo anterior, el código confirma dos condiciones: si el número es menor que 10 y si el número es igual o mayor que 10.

Para comprobar varias condiciones, Javascript ofrece la sentencia switch.

```
<script>
  var mivariable = 9;
  switch(mivariable) {
    case 5:
      alert("el número es cinco");
      break;
    case 8:
      alert("el número es ocho");
      break;
    case 10:
      alert("el número es diez");
      break;

    default:
      alert("el número es: " + mivariable);
  }
</script>
```

Código 4-12

Uso de la instrucción switch.

La sentencia switch evalúa una expresión (por lo general solo una variable), compara el resultado con las declaraciones de case que se indican entre llaves y, en caso de éxito, ejecuta las instrucciones declaradas para esa condición en concreto.

En el ejemplo del Código 4-12, switch evalúa la variable mivariable y a continuación compara su valor con cada caso. Si el valor es 5, por ejemplo, el control es transferido a la primera construcción case y el método alert() muestra el texto el número es cinco en la pantalla. Si este primer case no coincide con el valor de la variable, entonces se evalúa el siguiente caso, y así sucesivamente. Si no hay ninguna construcción case que coincida con el valor de la variable, se ejecutan las instrucciones de la etiqueta default.

Cada construcción case debe terminar con una instrucción con break, que informe al intérprete de Javascript, si se cumple la declaración, que no hay necesidad de seguir evaluando el resto de las declaraciones.

Las declaraciones switch e if son útiles pero realizan una tarea simple: evalúan una expresión, ejecutan un grupo de instrucciones de acuerdo con el resultado y después devuelven el control al script. Hay ciertas situaciones en las que esto no es suficiente. Puede que sea necesario ejecutar las instrucciones varias veces para una misma condición o volver evaluar la condición después de alguna modificación realizada durante el proceso. En estas situaciones podemos usar dos comandos: for y while.

```
<script>

  var mivariable = 9;

  for(var f = 0; f < mivariable; f++) {

    alert("El número actual es: " + f);

  }

</script>
```

Código 4-13

Uso de la instrucción for.

La declaración for, cuya sintaxis es for(inicio; condición; incremento), ejecuta el código entre llaves siempre que la condición declarada como el segundo parámetro sea verdadera. El primer parámetro establece los valores iniciales para el bucle, el segundo parámetro es la condición que debe ser contrastada y el último parámetro es la instrucción que determinará cómo se desarrollarán los valores iniciales en cada ciclo.

En el script del Código 4-13 se declara una variable llamada f y se le asigna 0 como valor de inicio. La condición es que el valor de la variable f sea menor que el valor de mivariable. Si se cumple la condición, se ejecuta el código entre llaves. Después de esto, el intérprete de Javascript ejecuta el último parámetro de for y comprueba la condición de nuevo. Si la condición es verdadera, se ejecutan las instrucciones una vez más. El bucle continúa hasta que la condición es falsa.

La variable f comienza con el valor 0. Después de que el primer ciclo ha terminado, la instrucción f++ aumenta el valor de f en 1 y la condición se comprueba de nuevo. En cada ciclo el valor de f se incrementa en 1. Este proceso continúa hasta que f alcanza un valor igual a 9, lo que hace que la condición falsa (9 no es menor que 9) y el bucle se interrumpe.

Conceptos básicos

El operador ++ es un operador Javascript para la suma. Añade al valor de la variable un valor igual a 1 y almacena el resultado en la misma variable. Es un atajo para la construcción variable = variable + 1.

La declaración for es útil cuando somos capaces de prever ciertas condiciones, tales como el valor inicial del bucle o cómo se desarrollarán esos valores. Cuando las condiciones no son tan claras se puede utilizar la instrucción while.

```
<script>
  var mivariable = 9;
  while(mivariable < 100) {
    mivariable++;
  }
  alert("El valor de mi variable ahora es: " + mivariable);
</script>
```

Código 4-14

Uso de la instrucción while.

La instrucción while solo requiere una declaración de la condición entre paréntesis y el código que se ejecutará entre llaves. El bucle se ejecuta hasta que la condición sea falsa. Si la primera evaluación de la condición devuelve el valor false, el código no se ejecuta. Si desea que las instrucciones se ejecuten al menos una vez, sin importar el resultado de la condición, entonces se utiliza la construcción do while.

```
<script>
  var mivariable = 9;
  do{
    mivariable++;
  }while(mivariable > 100);
  alert("El valor final de mivariable es: " + mivariable);
</script>
```

Código 4-15

Uso de la instrucción do while.

En el ejemplo del Código 4-15 cambiamos la condición por “mivariable es mayor que 100”. Dado que el valor inicial asignado a la variable era 9, la condición es falsa, pero como estamos usando la construcción do while, el código dentro del bucle se ejecuta una vez antes de que la condición sea evaluada. Al final, la ventana emergente muestra el valor 10 (porque 9 + 1 = 10).

Hasta el momento hemos venido trabajando en el espacio global. Éste es un espacio principal en el que se procesa el código a la vez que es leído por el intérprete. Las instrucciones se ejecutan secuencialmente en el orden en el que se enumeran en el código, pero nosotros podemos cambiar este orden mediante el uso de las funciones.

Una función es básicamente un bloque de código identificado por un nombre. Declarar el código dentro de una función en lugar de hacerlo en el espacio global ofrece varias ventajas y la principal es que aunque se declaren, no se ejecuta hasta que son llamadas por su nombre.

```
<script>
  function mifuncion(){
    alert("Esta es una función");
  }
  mifuncion();
</script>
```

Código 4-16

Declaración de funciones.

Conceptos básicos

En Javascript las comillas son intercambiables. Una cadena puede ser declarada con comillas dobles o simples (por ejemplo, tanto “Hola” como ‘Hola’). Si el texto de la cadena incluye el mismo tipo de comillas que se utilizó para declarar, entonces tenemos que marcar el carácter utilizando una barra invertida (por ejemplo, ‘I\’m Tom’). *

* (Nota del traductor: En este caso la barra invertida indica que el signo’ es un apóstrofe utilizado para el posesivo inglés y no la comilla de cierre de la cadena de texto.)

Las funciones se declaran con la palabra clave function, el nombre y el código entre llaves. Para llamar a una función (ejecutarla), solo es necesario utilizar su nombre con un par de paréntesis al final, como se muestra en el Código 4-16.

Importante

El propósito de este libro no es el estudio de funciones anónimas y patrones complejos de Javascript, pero vamos a ver otros ejemplos de patrones de Javascript y funciones anónimas en este capítulo y los siguientes.

Igual que sucede con las variables, el intérprete de Javascript lee la función, almacena su contenido en la memoria y asigna una referencia al nombre de la función. Cuando llamamos a la función, el intérprete comprueba la referencia y lee la función almacenada en la memoria. Esto nos permite llamar a la función una y otra vez, cada vez que sea necesario, lo que convierte a las funciones en las unidades de procesamiento de gran alcance.

```
<script>
  var mivariable = 5;
  function mifuncion(){
    mivariable = mivariable * 2;
  }
  for(var f = 0; f < 10; f++){
    mifuncion();
  }
  alert("El valor de mi variable ahora es: " + mivariable);
</script>
```

Código 4-17

Procesamiento de datos con funciones.

El ejemplo del Código 4-17 combina diferentes declaraciones ya estudiadas. En primer lugar se declara una variable y se le asigna el valor 5. Entonces, se declara (pero no se ejecuta) una función denominada mifuncion(). A continuación, se utiliza una instrucción para crear un bucle, que se ejecutará siempre que el valor de la variable f sea menor que 10. Y finalmente, el último valor almacenado en la variable mivariable se muestra en la pantalla. El código dentro del bucle llama a la función, por lo que mifuncion() se ejecutará en cada ciclo del bucle. Esta función multiplica el valor de mivariable por 2 y almacena el resultado en la misma variable. Cada vez que se ejecuta la función, el valor de mivariable se incrementa. Podrá darse cuenta del potencial de las funciones como unidades de procesamiento: se declaran una vez, se les da un nombre para una referencia y luego pueden ser utilizadas en cualquier momento.

También es posible utilizar una función anónima y asignarla a una variable. Para llamar a la función más adelante, habrá que llamar a la variable. Esta técnica se utiliza para definir funciones dentro de otras funciones, transformar funciones en constructores de objetos y crear complejos patrones de programación, como veremos más adelante.

```
<script>
  var mivariable = 5;
  var mifuncion = function(){
    mivariable = mivariable * 2;
  }
  for(var f = 0; f < 10; f++){
    mifuncion();
  }
  alert("El valor de mi variable ahora es: " + mivariable);
</script>
```

Código 4-18

Declaración de funciones con funciones anónimas.

Las funciones anónimas no tienen un nombre o identificador (de ahí su nombre). Podemos asignar estas funciones a variables, crear métodos para objetos, convertirlas en argumentos de otras funciones (o devolverlas desde otras funciones), etc. Las funciones anónimas son muy útiles en Javascript y sin este tipo de construcciones, los patrones complejos de programación no serían posibles.

Las variables pueden tener diferentes ámbitos y en los ejemplos anteriores se utiliza una variable definida en el espacio global. Los valores definidos en el espacio global tienen alcance global, es decir, pueden ser utilizados en cualquier parte del código. Sin embargo, las variables declaradas dentro de otras funciones tienen un ámbito local, lo que significa que solo pueden usarse dentro de la función en la que han sido declaradas. Ésta es otra de las ventajas de las funciones: permiten almacenar información privada a la que no podrá accederse desde otras partes del código. Esto ayuda a evitar duplicados que puedan dar lugar a errores o sobrescribir accidentalmente variables y valores. Si hay dos variables con el mismo nombre, pero una está en el espacio global y otra dentro de una función, se considerarán como dos variables diferentes.

```
<script>
  var mivariable = 5;
  function mifuncion(){
   var myvariable = "Esta es una variable de función";

    alert(mivariable);
  }
  mifunction();
  alert(mivariable);
</script>
```

Código 4-19

Declaración de variables locales y globales.

En el script anterior se declaran dos variables llamadas mivariable: una en el espacio global y otra dentro de la función mifuncion(). También hay dos alert() en el código: uno dentro de la función y otro en el espacio global, al final del código. Ambos métodos muestran el contenido de mivariable, pero, como verá si ejecuta el script en su navegador, son variables diferentes. La variable dentro de la función hace referencia a un espacio en la memoria que contiene el texto Ésta es una una variable de función, mientras que la variable en el espacio global (declarada antes) hace referencia a un espacio en la memoria que contiene el valor 5.

Importante

También es posible crear variables globales desde funciones. Omitir la palabra clave var cuando se declara la variable dentro de una función es suficiente para que la variable sea global.

Las variables globales son útiles cuando las funciones tienen que compartir valores comunes, porque es posible acceder a ellas desde cualquier función en el script. Pero podemos sobrescribir accidentalmente sus valores o borrarlos desde alguna instrucción del código, o incluso desde otros códigos, pues todos los códigos del documento comparten el mismo espacio global y por eso se recomienda evitar su uso siempre que sea posible.

Conceptos básicos

Una práctica común para evitar el uso de variables globales es declarar un objeto global y luego crear la aplicación dentro de este objeto. Vamos a estudiar los objetos en breve.

Importante

Estudiaremos más aspectos de las funciones y practicaremos todas estas técnicas más adelante.

Es posible leer variables globales desde una función, pero no es posible hacer lo contrario. Por este motivo Javascript ofrece diferentes opciones de comunicación entre una función y el espacio global. Podemos enviar los datos a la función a través de sus argumentos o podemos enviar los datos desde la función a través de la instrucción return.

```
<script>
 var mivariable = 5;
 var miresultado;
 function mifuncion(contar){
   contar = contar + 12;
   return contar;
 }
 miresultado = mifuncion(mivariable);
 alert(miresultado);
</script>
```

Código 4-20

Comunicación mediante funciones.

Los argumentos son variables que están definidas entre los paréntesis de la función y se pueden utilizar dentro de la función como cualquier otra variable. Toman los valores enviados por el código al llamar a la función. En el Código 4-20, la función mifuncion() se declara con un argumento llamado contar. Más tarde, se llama la función con la variable mivariable entre paréntesis. Éste es el valor que recibirá la función y asignará a contar.

Cuando se invoca la función, el valor inicial del contador es 5 (porque ése es el valor inicial declarado por mivariable). Dentro mifuncion(), se suma el valor 12 a contar y finalmente return devuelve el resultado. De nuevo en el espacio global, se asigna a la variable miresultado el valor devuelto por la función (que como recordará era 17). Finalmente el contenido de esa variable es mostrado en la pantalla.

Tenga en cuenta que al principio se ha declarado la variable miresultado pero no se le ha asignado un valor. Esto no solo es perfectamente aceptable sino que además es recomendable, y en general es aconsejable declarar desde el comienzo todas las variables con las que se va a trabajar, para evitar confusiones y después poder identificar a cada una desde otras partes del código.

4.1.4 Objetos

Los objetos son como grandes variables capaces de contener otras variables (llamadas propiedades), así como funciones (llamadas métodos). Hay distintas formas de declarar objetos, pero la más simple es utilizar una notación literal.

```
<script>
    var miobjeto = {
        nombre: "Juan",
        edad: 30
    }
</script>
```

Código 4-21

Creación de objetos.

Los objetos se declaran usando la palabra clave var y un par de llaves. Las propiedades y los métodos requieren un nombre y un valor. Se declaran entre las llaves utilizando dos puntos después del nombre y una coma para separar cada declaración (la última declaración no requiere una coma).

En el ejemplo del Código 4-21, declaramos el objeto miobjeto con dos propiedades: nombre y edad. El valor de la propiedad nombre es Juan y el valor de la propiedad edad es 30. A diferencia de lo que sucede con las variables, no podemos acceder a los valores de las propiedades utilizando simplemente su nombre. En primer lugar debemos especificar el objeto al que pertenecen y hay dos formas de hacerlo: utilizando la notación de puntos o mediante corchetes.

```
<script>
  var miobjeto = {
    nombre: "Juan",
    edad: 30
  }
  alert("Mi nombre es " + miobjeto.nombre);
  alert("Yo tengo " + miobjeto['edad'] + " años");
</script>
```

Código 4-22

Acceder a las propiedades.

En el Código 4-22 usamos dos anotaciones para acceder a las propiedades del objeto. En pantalla se obtienen dos mensajes de alerta: Mi nombre es Juan y Tengo 30 años. El uso de la notación es irrelevante, excepto en algunas circunstancias. Por ejemplo, cuando hay que acceder a la propiedad a través del valor de una variable, es necesario usar corchetes.

```
<script>
  var mivariable = "nombre";
  var miobjeto = {
    nombre: "Juan",
    edad: 30
  }
  alert(miobjeto[mivariable]);
</script>
```

Código 4-23

Acceder a las propiedades utilizando variables.

En el Código 4-23 no hubiera sido posible acceder a la propiedad con una notación de puntos (miobjeto.mivariable) porque el intérprete habría intentado encontrar una propiedad llamada mivariable, que no existe. Al usar corchetes, primero se resuelve la variable y luego se accede al objeto con el valor de la variable (nombre) en lugar de su nombre. El resultado mostrado en pantalla es Juan.

También es necesario acceder a la propiedad mediante corchetes cuando su nombre se considera inválido para una variable (porque incluye caracteres no válidos, un espacio, o comienza con un número). En el ejemplo siguiente, el objeto incluye una propiedad declarada con un texto como un nombre. Se puede declarar los nombres de propiedades como textos entre comillas, pero como este texto en particular tiene un espacio, el código miobjeto.mi edad daría un error. Por este motivo hay que usar corchetes para acceder a la propiedad.

```
<script>
  var mivariable = "nombre";
  var miobjeto = {
    nombre: "Juan",
    'mi edad': 30
  }
  alert(miobjeto['mi edad']);
</script>
```

Código 4-24

Propiedades con nombres inválidos.

Además de leer el valor de las propiedades, también es posible establecerlos o modificarlos utilizando las dos notaciones. En el siguiente ejemplo modificaremos el valor de la propiedad nombre y añadiremos una nueva propiedad denominada empleo. De modo que el objeto tendrá tres propiedades: nombre, edad y empleo. Al final de la secuencia de comandos, se muestran todos los valores en la pantalla.

```
<script>
  var miobjeto = {
    nombre: "Juan",
    edad: 30
  }
  miobjeto.nombre = "Martín";

  miobjeto.empleo = "Programador";
  alert(miobjeto.nombre + " " + miobjeto.edad + " " + miobjeto.
empleo);
</script>
```

Código 4-25

Actualizar de los valores y añadir nuevas propiedades al objeto.

Como hemos mencionado antes, los objetos también pueden incluir funciones. Las funciones dentro de un objeto se denominan métodos y convierten al objeto en una unidad capaz de contener y procesar información de forma independiente.

```
<script>
  var miobjeto = {
    nombre: "Juan",
    edad: 30,
    mostrarnombre: function(){
        alert(miobjeto.nombre);
    },
    cambiarnombre: function(nuevonombre){
      miobjeto.nombre = nuevonombre;
    }
  }
  miobjeto.mostrarnombre();
  miobjeto.cambiarnombre('''Jonatan');
  miobjeto.mostrarnombre();
</script>
```

Código 4-26

Incluir funciones dentro del objeto.

La secuencia de comandos del Código 4-26 muestra no solo cómo incluir métodos en un objeto, sino también cómo utilizarlos. Los métodos tienen la misma sintaxis que las propiedades: requieren dos puntos después del nombre y una coma para separar cada declaración. Se diferencian de las propiedades en que los métodos no tienen valores directos, sino que se construyen utilizando funciones anónimas.

En el ejemplo anterior se crean dos métodos, mostrarnombre() y cambiarnombre(). El método mostrarnombre() muestra una ventana emergente con el valor de la propiedad nombre y el método cambiarnombre () asigna a la propiedad nombre el valor recibido del atributo nuevonombre. Se trata de dos métodos independientes que trabajan en la misma propiedad. Para ejecutar los métodos, se utiliza la notación de puntos y paréntesis después del nombre. Al igual que las funciones, los métodos pueden tomar atributos y devolver valores usando la declaración return. En este ejemplo, cambiarnombre() requiere un atributo con el nuevo nombre.

Los objetos también pueden contener otros objetos y la estructura se puede hacer más y más compleja de acuerdo con las necesidades necesidades. Para acceder a las propiedades y métodos de esos objetos internos, hay que concatenar sus nombres utilizando la notación de punto.

```
<script>
  var miobjeto = {
    nombre: "Juan",
    edad: 30,
    moto: {
      modelo: "Suzuki",
      año: 1981
    },
    mostrarvehiculo: function(){
      alert("La " + miobjeto.moto.modelo + " pertenece a " +
 miobjeto.nombre);
    }
  }
  miobjeto.mostrarvehiculo();
</script>
```

Código 4-27

Creación de objetos dentro de objetos.

Como podrá ver, los objetos son unidades de procesamiento independientes y de gran alcance. Son tan importantes en Javascript que el lenguaje no solo ofrece diferentes alternativas para crear sus propios objetos sino que también ofrece su propia cuenta. De hecho, casi todo en Javascript es un objeto. Podemos usar objetos nativos, crear nuestros propios objetos desde cero, y modificarlos o expandirlos. Pero una de las características más importantes de los objetos es la posibilidad de crear otros nuevos objetos que heredan las propiedades y métodos de los objetos con los que han sido creados.

En Javascript, la herencia (la forma en la que los objetos obtienen las mismas propiedades y métodos de otros objetos) se realiza a través de prototipos. Cada objeto tiene un prototipo y a partir de ese prototipo se crean los nuevos objetos.

El objeto miobjeto de ejemplo anterior adquiere automáticamente su propio prototipo cuando es creado. Si decidimos crear un objeto nuevo basado en miobjeto, este nuevo objeto se creará a partir del prototipo de miobjeto, y no desde el mismo objeto. En la Figura 4-1, el objeto empleado se crea a partir del prototipo miobjeto. Pero este nuevo objeto también tiene su propio prototipo. Si más tarde se crean nuevos objetos a partir del objeto empleado, se basarán en el prototipo empleado. Esta construcción puede ser tan larga como sea necesario y crea una cadena de objetos llamada cadena de prototipos. Las modificaciones introducidas en un prototipo afectarán a todos los objetos de la cadena. Si introducimos cambios en el prototipo miobjeto, por ejemplo, empleado y todos los objetos basados en el prototipo empleado heredarán estos cambios.

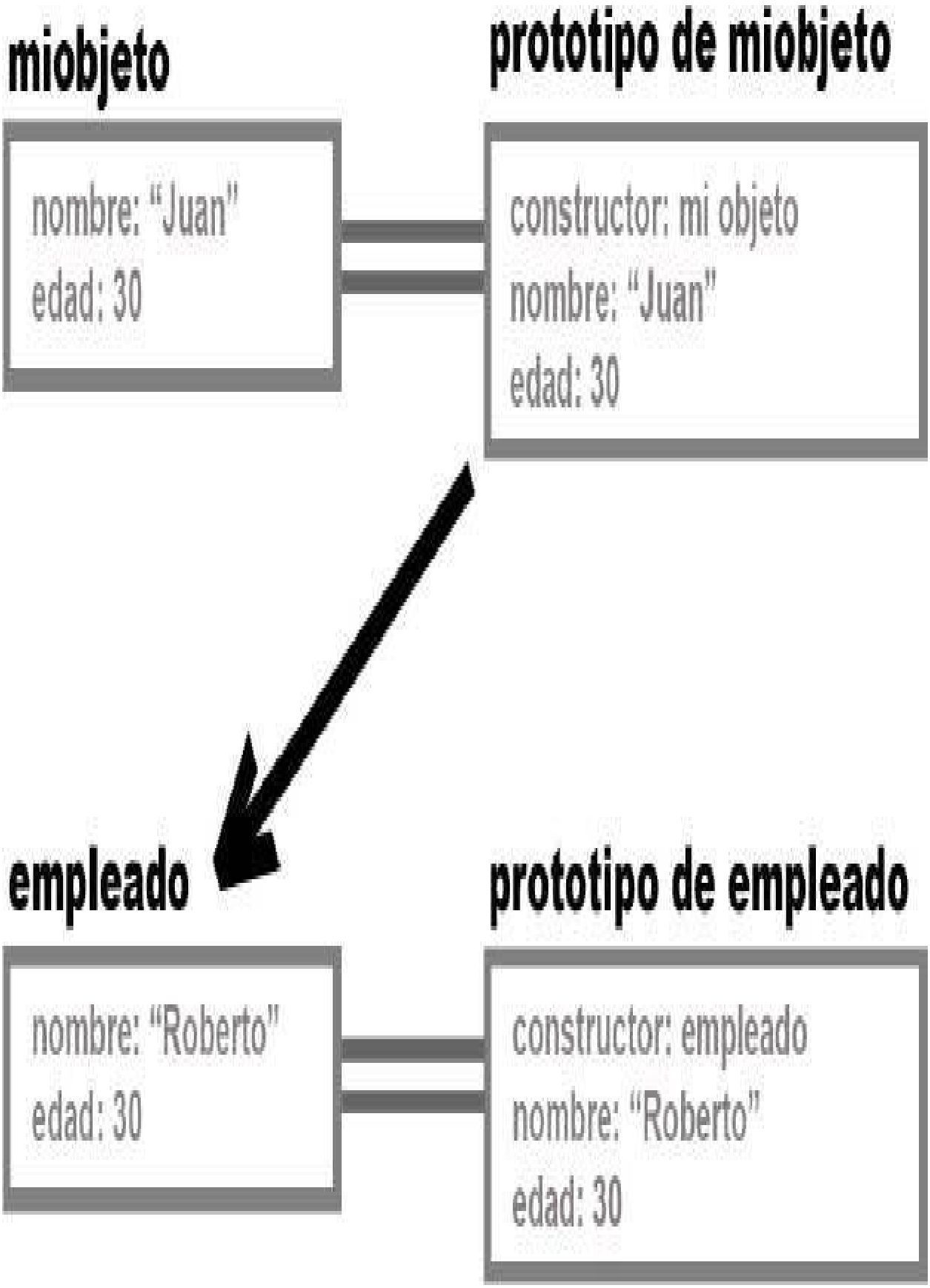

miobjeto
nombre: "Juan"
edad: 30
prototipo de miobjeto
constructor: mi objeto
nombre: "Juan"
edad: 30
empleado
nombre: "Roberto"
edad: 30
prototipo de empleado
constructor: empleado
nombre: "Roberto"
edad: 30

Figura 4-1

Herencia de prototipos.

En otros lenguajes orientados a objetos, las técnicas disponibles para lograr este tipo de herencia eran desordenadas e inconsistentes y esto disuadió a algunos programadores de trabajar con Javascript por un tiempo. El nuevo método create() acabó con esto. Este método es parte de un objeto nativo llamado Object y utiliza un objeto existente como prototipo de uno nuevo, de manera que ya no hay que preocuparse por cómo trabajar con los prototipos.

```
<script>
  var miobjeto = {
    nombre: "Juan",
    edad: 30,
    mostrarnombre: function(){
        alert(this.nombre);
    },
    cambiarnombre: function(nuevonombre){
      this.nombre = nuevonombre;
    }
  }

  var empleado = Object.create(miobjeto);
  empleado.cambiarnombre('Roberto');
  empleado.mostrarnombre();
  miobjeto.mostrarnombre();
</script>
```

Código 4-28

Generación de nuevas instancias.

Es importante observar cómo hemos hecho referencia al objeto con el que se está trabajando en el Código 4-28: y es que los objetos pueden referirse a ellos mismos usando la palabra clave this.

En este ejemplo utilizamos this en lugar del nombre del objeto (miobjeto) para tener acceso a la propiedad de nombre desde el interior del objeto. La palabra clave declara “este objeto” en lugar de proporcionar el nombre del objeto. Se usa this porque este objeto sirve como un modelo para la creación de otros nuevos objetos o instancias. Estas copias tendrán su propio nombre, distinto al del objeto original.

El uso de this es necesario en casos como éste, en el que se quiere representar el nombre del objeto que se está trabajando, independientemente de que se trate del objeto original o de una de las copias. Por ejemplo, en el objeto empleado, creado a partir miobjeto en la secuencia de comandos del Código 4-28, la palabra clave this representa el objeto empleado, por lo que la construcción this.nombre en este objeto devolverá el mismo valor que empleado.nombre.

El objeto empleado se crea utilizando el método create(), que solo requiere el nombre del objeto que hará de prototipo para el nuevo y devuelve un nuevo objeto que puede ser asignado a una variable.

En el Código 4-28, la instancia se crea primero con Object.create() y luego se le asigna la variable empleado. Una vez que se ha creado la instancia, se pueden actualizar sus valores. Utilizando el método cambiarnombre() se cambia el nombre del empleado por Roberto y luego muestra el valor de cada propiedad nombre en la pantalla.

Importante

create() es un nuevo método introducido en las versiones recientes del lenguaje. Para emular este método en los navegadores antiguos, Douglas Crockford recomienda en su sitio web (www.crockford.com) y en su libro Javascript: The Good Parts (publicado por O'Reilly Media / Press Yahoo), el uso de la secuencia de comandos siguiente:

```
if(typeof Object.create !== 'function') {
    Object.create = function(o) {
        function F() {}
        F.prototype = o;
        return new F();
    };
}
```

Ahora tenemos dos objetos individuales, miobjeto y empleado, con sus propias propiedades, métodos y valores, pero vinculados a través de la cadena de prototipos. El nuevo objeto empleado no es solo una copia, es una instancia, un objeto que mantiene un vínculo con su prototipo. Cuando el prototipo cambia, las instancias heredan esos cambios:

```
<script>
  var miobjeto = {
    nombre: "Juan",
    edad: 30,
    mostrarnombre: function(){
        alert(this.nombre);
    },
    cambiarnombre: function(nuevonombre){
      this.nombre = nuevonombre;
    }
  }

  var empleado = Object.create(miobjeto);
  empleado.edad = 24;
  miobjeto.mostraredad = function(){
    alert(this.edad);
  };
  empleado.mostraredad();
</script>
```

Código 4-29

Adición de un nuevo método para el prototipo.

En el Código 4-29, se crea el objeto empleado utilizando miobjeto como prototipo, como lo hemos hecho antes. Luego se cambia el valor de la propiedad edad de este nuevo objeto a 24. Y, por último, se añade un método llamado mostraredad() al prototipo de objeto miobjeto.

Gracias a la cadena de prototipos, este nuevo método es ahora accesible desde las instancias. Por lo tanto. cuando llamamos al método mostraredad() del objeto empleado al final del script, el intérprete busca el método en el objeto empleado y luego en toda la cadena de prototipos, hasta que finalmente lo encuentra en miobjeto. Cuando el método es finalmente encontrado, se muestra el valor 24 en la pantalla. Esto se debe a que, aun cuando el método mostraredad() pertenece a miobjeto, la palabra clave this en este método apunta al objeto con el que estamos trabajando (empleado). Gracias a la cadena de prototipo es posible invocar el método mostraredad() desde empleado, y gracias a la palabra clave this, la propiedad de edad que se muestra en pantalla es la que pertenece justamente a empleado.

El método create() es tan simple como potente. Se toma un objeto y lo convierte en el prototipo de uno nuevo. Esto nos permite construir una cadena de objetos en la que cada uno hereda las propiedades y métodos de sus predecesores. En el siguiente ejemplo vamos a demostrar el uso de este atributo, creando un total de cuatro objetos.

El objeto inicial miobjeto es el prototipo de empleado1, pero luego usamos

empleado1 como prototipo para empleado2 y empleado2 como prototipo para empleado3. Todos los objetos son independientes, pero están conectados el uno al otro por la cadena de prototipos. Cuando se añaden propiedades o métodos a uno de los objetos (un prototipo), el resto de los objetos de la cadena también tiene acceso a ellos.

```
<script>
  var miobjeto = {
    nombre: "Juan",
    edad: 30,
    mostrarnombre: function(){
        alert(this.nombre);
    },
    cambiarnombre: function(nuevonombre){
      this.nombre = nuevonombre;
    }
  }
  var empleado1 = Object.create(miobjeto);
  var empleado2 = Object.create(empleado1);
  var empleado3 = Object.create(empleado2);
  empleado2.mostraredad = function(){
    alert(this.edad);
  };
  empleado3.edad = 24;
  empleado3.mostraredad();
</script>
```

Código 4-30

Prueba de la cadena de prototipos.

En el Código 4-30 se demuestra la cadena de prototipo añadiendo el método mostraredad() a empleado2. Ahora empleado2 y empleado3 (y también cualquier objeto creado en base a más de estos dos objetos) tienen acceso a este método, pero si se intenta llamar al método en empleado1, no va a funcionar. Esto se debe a que la herencia de la cadena no funciona hacia arriba.

A pesar de la simplicidad de Object.create(), a veces tiene que ser complementada con complejos patrones y funciones de apoyo para permitir que ciertas características en el código como la creación dinámica de objetos, valores de inicialización, y las propiedades y métodos privados.

Dependiendo de las necesidades, puede ser mejor para crear los objetos con una técnica alternativa que combina la notación literal, funciones anónimas y la instrucción return.

```
<script>
  var constructor = function(nombreinicial){
    var obj = {
      nombre: nombreinicial,
      edad: 30,
      mostrarnombre: function(){
          alert(this.nombre);
      },
      cambiarnombre: function(nuevonombre){
        this.nombre = nuevonombre;
      }
    };
    return obj;
  };
  empleado = constructor("Juan");
  empleado.mostrarnombre();
</script>
```

Código 4-31

Uso de funciones anónimas para crear objetos.

En el ejemplo del Código 4-31, se utiliza una función anónima para crear un objeto. La función anónima se asigna a la variable constructor y a través de esta variable se llama a la función con el valor inicial Juan. Este valor es tomado por el atributo de función nombreinicial y se asigna a la propiedad nombre del objeto. Por último, el objeto llamado obj es devuelto por la declaración return y asignado a la variable empleado.

Usando esta técnica es posible crear nuevos objetos de forma dinámica, proporcionar valores iniciales o incluso realizar algún procesamiento en el comienzo de la función y, a continuación, devolver un objeto creado por este procesamiento.

Como puede ver, es más que la creación y duplicación de un objeto. La ventaja más importante de esta técnica es la posibilidad de definir propiedades y métodos privados.

Todos los objetos e instancias que hemos creado hasta ahora tienen propiedades y métodos públicos, lo que significa que se puede acceder y modificar el contenido de los objetos desde cualquier parte del código. Para hacer que las propiedades y métodos estén disponibles solo para el objeto que los ha creado, tenemos que hacerlos privados usando una técnica que llamaremos cierre.

Como explicamos antes, las variables creadas en el espacio global son accesibles desde cualquier parte del código, mientras que las variables creadas dentro de las funciones solo son accesibles a partir de estas funciones en las que han sido creadas. Lo que no hemos mencionado aún, es que funciones y métodos mantienen un vínculo con el entorno en el que han sido creados y con las variables en ellas creadas. Cuando se devuelve un objeto a partir de una función, sus métodos son capaz de acceder a las variables de la función, incluso cuando éstas ya no se encuentran en el mismo ámbito.

A continuación estudiaremos el comportamiento de las funciones que acabamos de explicar en un nuevo código de ejemplo.

```
<script>
  var constructor = function(){
    var nombre = "Juan";
    var edad = 30;
    var obj = {
      mostrarnombre: function(){
          alert(nombre);
      },
      cambiarnombre: function(nuevonombre){
        nombre = nuevonombre;
      }
    };
    return obj;
  };
  empleado = constructor();
  empleado.mostrarnombre();
</script>
```

Código 4-32

Creación de las propiedades privadas de un objeto.

La secuencia de comandos en el Código 4-32 es exactamente la misma que la del ejemplo anterior excepto que en lugar de declarar nombre y edad como propiedades del objeto, se declaran como variables de la función que devuelve el objeto. El objeto devuelto recordará estas variables, pero será el único que tiene acceso a ellas. No hay manera de modificar el valor de esas variables desde otros elementos en el otro código salvo utilizar los métodos del objeto devuelto (en este caso, mostrarnombre() y cambiarnombre()). Esto se denomina cierre. La función está cerrada y su entorno deja de ser accesible, pero el elemento sigue unido a él (un objeto en nuestro ejemplo).

Hágalo usted mismo

Cree nuevos objetos a partir de la función del Código 4-32 y utilice los métodos cambiarnombre() y mostrarnombre() para acceder a sus propiedades. Por ejemplo, empleado2 = miobjeto();.

Por supuesto, cada nueva instancia creada a partir de esta función tendrá la misma estructura y tendrá acceso a sus propias propiedades privadas.

4.1.5 Constructores

Las funciones utilizadas para crear objetos en los ejemplos anteriores se llaman constructores. Javascript incluye una clase especial de constructor que trabaja con el operador new para obtener un nuevo objeto. Este método no es parte de un lenguaje prototípico pero se introdujo en Javascript para mantener la consistencia con otros lenguajes orientados a objetos.

Este tipo de constructores no devuelven un objeto, sino que son los planos a partir de los cuales se crean los objetos. Por eso requieren el uso de la palabra clave this para declarar o acceder a propiedades y métodos.

```
<script>
  function Miobjeto(){
    this.nombre = "Juan";
    this.edad = 30;
    this.mostrarnombre = function(){
        alert(this.nombre);
    };
    this.cambiarnombre = function(nuevonombre){
      this.nombre = nuevonombre;
    };
  }
  var empleado = new Miobjeto();
  empleado.mostrarnombre();
</script>
```

Código 4-33

Uso de constructores.

Excepto por el uso de this, los constructores y las funciones tienen casi el mismo aspecto. Recomendamos comenzar su nombre con mayúscula, como en el ejemplo del Código 4-33, para indicar la presencia de los constructores.

Para crear un objeto de un constructor que tenemos que utilizar el operador new. Este operador toma el nombre del constructor y devuelve un objeto. El constructor puede incluir atributos para proporcionar valores iniciales al nuevo objeto.

```
<script>
  function Miobjeto(nombreinicial, edadinicial){
    this.nombre = nombreinicial;
    this.edad = edadinicial;
    this.mostrarnombre = function(){
        alert(this.nombre);
    };
    this.cambiarnombre = function(nuevonombre){
      this.nombre = nuevonombre;
    };
  }
  var empleado = new Miobjeto("Roberto", 55);
  empleado.mostrarnombre();
</script>
```

Código 4-34

Definición de valores iniciales para el objeto.

Este constructor no transforma un objeto en prototipo de otro, como lo hace el método create(). Más bien crea nuevas instancias a partir del prototipo del constructor (Miobjeto.prototype). Por este motivo, si queremos introducir modificaciones que puedan afectar a las instancias, no podemos trabajar directamente con el objeto inicial como lo hicimos antes. Tenemos que acceder prototipo del constructor mediante la propiedad prototype.

```
<script>
  function Miobjeto(nombreinicial, edadinicial){
    this.nombre = nombreinicial;
    this.edad = edadinicial;
    this.mostrarnombre = function(){
        alert(this.nombre);
    };
    this.cambiarnombre = function(nuevonombre){
      this.nombre = nuevonombre;
    };
  }
  Miobjeto.prototype.mostraredad = function(){
    alert(this.edad);
  };
  var empleado = new Miobjeto("Roberto", 55);
  empleado.mostraredad();
</script>
```

Código 4-35

Aumentar el prototipo.

El constructor del Código 4-35 es el mismo que el del ejemplo anterior. Aquí se añade al prototipo de miobjeto un nuevo método llamado mostraredad(). A partir de ahora, todas las instancias incluirán este nuevo método.

Importante

Los prototipos son la esencia de Javascript, pero son difíciles de asimilar. Si se acostumbra a trabajar con el método create() y la notación literal que estudió en el comienzo de este capítulo, en la mayoría de los casos no será necesario el uso de prototipos, constructores y de la palabra clave new. De hecho, la forma ideal de trabajar con Javascript es usando el método create() para crear objetos e instancias sin entrar en la complejidad de su naturaleza prototípica, sin embargo, estas técnicas son necesarias en determinadas circunstancias, y las librerías elementales y API exigen constantemente su uso, como veremos más adelante, así que es bueno que esté familiarizado con ellas.

Las opciones que ofrece Javascript en este aspecto son casi ilimitadas. Para obtener más información acerca de los patrones de Javascript y cómo trabajar con los objetos, por favor visite nuestro sitio web y siga los enlaces de este capítulo.

4.1.6 El objeto Window

Cada vez que se inicia el explorador, se crea un objeto global llamado Window para hacer referencia a la ventana del navegador y proporcionar algunos métodos y propiedades esenciales. Todo está incluido dentro de este objeto, desde los códigos hasta el documento entero.

El objeto Window es accesible desde Javascript a través de la propiedad Window. Esta propiedad debería ser mencionada cada vez que queramos acceder a las propiedades y métodos de Javascript, aunque no es estrictamente necesario. Puesto que Window es el objeto global, el intérprete infiere que cualquier cosa que no tiene una referencia, pertenece al objeto Window. Por ejemplo, podemos escribir el método alert() tal como lo hicimos en los ejemplos anteriores o mediante la fórmula Window.alert(). A continuación le presentamos una lista de las propiedades y métodos más comunes ofrecidos por este objeto.

alert(texto): Este método muestra una ventana pop-up en la pantalla con el valor entre paréntesis (ver ejemplos anteriores de este capítulo).

confirm(texto): Este método es similar a alert(), pero ofrece al usuario la posibilidad de escoger entre dos botones, sí y no. Devuelve true o false, de acuerdo con la respuesta del usuario (vea el ejemplo en el Capítulo 14, el Código 14-6).

setTimeout(función, millisegundos): Este método ejecuta la función especificada en el primer atributo al transcurrir el tiempo especificado en milisegundos por el segundo atributo. Es posible asignar este método a una

variable y luego cancelar el proceso utilizando clearTimeout() y el nombre de la variable entre paréntesis (por ejemplo, var tiempo = setTimeout(function(){alert(“Hola”);}, 5000);). Un ejemplo de la aplicación de este método se proporciona en el Capítulo 11, Código 11-13.

setInterval(function, milliseconds): Este método es similar a setTimeout() pero llama a la función repetidamente con la frecuencia del tiempo establecido por el segundo atributo, hasta que el proceso se cancela por clearInterval() (por ejemplo, var tiempo = setInterval(function() {alert(“Hola”);}, 2000);). Proporcionamos algunos ejemplos de este método en los capítulos 6 y 10.

location: Este objeto tiene varias propiedades para devolver información acerca de la dirección URL del documento actual. Puede también ser utilizado como una propiedad para establecer o devolver la dirección URL del documento (por ejemplo, window.location = “http://www.minkbooks.com”).

history: Este objeto proporciona diferentes métodos para navegar a través de la historia de la ventana. Esto incluye métodos como back(), para cargar el documento anterior, forward(), para cargar un documento más reciente de la lista, y go(), para cargar cualquier documento en la historia de la ventana abierta. Vamos a estudiar estos métodos junto con la API History en el Capítulo 18 de este manual.

4.1.7 El objeto Document

Ya hemos mencionado antes que casi todo en Javascript es un objeto, desde los objetos personalizados que creamos en los ejemplos de este capítulo hasta funciones, matrices, textos o números. Así mismo, todo el documento HTML y cada uno de los elementos HTML son objetos también.

En un entorno de navegador, se crea una estructura interna para procesar el documento HTML. La estructura de la que hablamos se llama DOM (del inglés Document Object Model) y se compone de objetos que representan cada elemento HTML. El DOM se inicia con el objeto Document, accesible desde Javascript a través de la propiedad document, que permite acceder o modificar cualquier elemento HTML del documento.

El objeto Document es un miembro del objeto Window y proporciona métodos y propiedades para trabajar con el documento. Vamos a estudiar y aplicar algunos de estos métodos más adelante en este capítulo.

4.2 Una introducción a los eventos

En Javascript, las acciones del usuario se denominan eventos. Cuando el usuario realiza una acción, como hacer clic con el ratón o pulsar una tecla, se activa un evento que es específico para cada acción. Además de los eventos producidos por un usuario, también hay eventos activados por el sistema. Un ejemplo es el evento load que se activa cuando el documento se ha cargado completamente. Estos eventos son manejados por los códigos o funciones enteras. El código que responde al evento se llama al controlador de eventos. Cuando se registra un controlador de eventos, se define cómo la aplicación va a responder al evento. La estandarización del método addEventListener() cambió la forma en la que se llama al procedimiento: en lugar de registrar un controlador del evento se añade un detector o escucha de eventos. Esto nos deja con tres diferentes maneras de registrar un controlador de eventos para un elemento HTML: añadir un nuevo atributo de evento al elemento, registrar un controlador de eventos con una propiedad de evento, o utilizar el nuevo método estándar addEventListener().

Importante

Aunque ahora presentamos solo teoría, veremos ejemplos de cómo aplicar estas técnicas en la siguiente parte de este capítulo.

4.2.1 Atributos de eventos

HTML proporciona atributos para insertar código Javascript en nuestros documentos. Estos atributos están asociados con eventos que ejecutan código de acuerdo con la respuesta del usuario o el estado actual del documento y los elementos. Los atributos se denominan atributos de eventos y sus valores son códigos Javascript que controlan los eventos (por ejemplo, <p onclick=“alter(‘Hola’)”> Pulsa aquí<p>). Los nombres de los atributos de eventos se hacen añadiendo el prefijo on al nombre del evento. Por ejemplo, el nombre del atributo de evento para el evento click es onclick. Los atributos de los eventos más utilizados son aquellos relacionados con el ratón, como onclick, onMouseOver o onMouseOut. Sin embargo, existen también atributos de eventos que responden a eventos de teclado y de ventana, y ejecutan una acción después pulsar una tecla o después de un cambio en el estado del documento (por ejemplo, onload o onfocus). He aquí una lista de los atributos de los eventos más comunes:

onLoad: Este atributo controla el evento load, que se activa cuando el documento se ha cargado.

onClick: Controla el evento click, que se activa cuando el usuario hace clic en el ratón.

onMouseOver: Controla el evento mouseover, que se activa cuando el puntero del ratón se mueve sobre el elemento.

onMouseOut: Controla el evento mouseout, que se activa cuando el puntero

del ratón se desplaza fuera del elemento.

onkeypress: Controla el evento keypress, que se activa cuando se pulsa una tecla y se suelta.

onkeydown: Controla el evento keyDown, que se activa cuando se pulsa una tecla cualquiera.

onkeyup: Controla el evento keyUp, que se activa cuando se suelta una tecla.

onChange: Controla el evento change, que se activa cuando cambia un elemento (normalmente un campo de entrada en un formulario).

Importante

HTML5 presenta varios nuevos eventos que estudiaremos en los capítulos siguientes junto con su API correspondiente. Para obtener una lista completa de los eventos tradicionales, por favor visite nuestro sitio web y siga los enlaces de este capítulo.

4.2.2 Propiedades del evento

Las propiedades de eventos son los mismos atributos de eventos pero se aplican en el código como las propiedades de los elementos. Utilizando los selectores de Javascript, es posible hacer referencia a un elemento HTML y asignarle a ese elemento la propiedad de evento adecuada (por ejemplo, element.onclick = mifuncion;).

Antes de HTML5, esta era la única técnica está disponible para registrar eventos en Javascript en diferentes navegadores. Algunos proveedores de navegadores estaban desarrollando sus propios sistemas, pero nada cuajó hasta que fue adoptado el nuevo estándar addEventListener().

Vamos a ver algunos ejemplos de propiedades de evento en la siguiente parte de este capítulo.

4.2.3 El método addEventListener()

El método addEventListener() es la técnica ideal para detectar eventos y fue implementada como estándar por HTML5. Utiliza la sintaxis addEventListener(evento, función, captura). El primer atributo es el nombre del evento (sin el prefijo), el segundo atributo es una función que responde al evento, usualmente llamada detector o escucha (o listener en inglés), que podría ser referencia de una función o una función anónima entera), y el tercer atributo establece, con true (verdadero) o false (falso), la forma en la que se ejecutarán múltiples eventos en el árbol HTML. El tercer atributo puede ser útil para la depuración o en otras circunstancias en las que tenemos que propagar el evento ejecutado por un elemento a todos los elementos superiores del árbol, pero por lo general el valor false es suficiente para la mayoría de las situaciones y por eso es el valor utilizado por defecto.

Aunque los resultados de la aplicación de esta técnica y la anterior son similares, addEventListener() nos permite añadir tantos detectores de eventos como queramos para el mismo elemento. Esta distinción da a addEventListener() una ventaja sobre otros métodos y lo convierte en la técnica ideal para aplicaciones HTML5.

Importante

Javascript también ofrece el método removeEventlis-tener() para eliminar un detector de un elemento. Este método requiere los mismos atributos que addEventListener().

Consulte el Código 3 del Capítulo 12 para ver un ejemplo de cómo implementar este método.

4.3 Incorporar Javascript

Igual que sucede con CSS, hay tres diferentes técnicas para incorporar código Javascript en un documento HTML: en línea, incrustado y desde archivos externos.

4.3.1 En línea

La técnica en línea saca provecho de los atributos de eventos estudiados previamente. Ésta es una técnica en desuso pero todavía útil y práctica en algunas circunstancias. Para que un elemento de responda a un evento, solo hace falta agregar el atributo correspondiente con el código que lo controla.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Javascript</title>
</head>
<body>
  <section id="principal">
    <p onclick="alert('¡Has hecho clic!')">Pulsa aquí</p>
    <p>Aquí no puedes pulsar</p>
  </section>
</body>
</html>
```

Código 4-36

Código Javascript en línea.

En el Código 4-36, gracias el atributo de evento onclick se ejecuta un código Javascript que se inicia cada vez que el usuario hace clic en el texto Pulsa aquí. El atributo onclick dice algo así como: “cuando alguien hace clic en este elemento se ejecuta este código,” y el código es, en este caso, la función predefinida de Javascript alert(), que muestra una pequeña ventana con el mensaje ¡Has hecho clic!. Si cambia el atributo onclick por onMouseOver, por ejemplo, y el código se ejecutará con solo situar el puntero del ratón sobre el elemento.

Hágalo usted mismo

Copie el Código 4-36 y los siguientes de este capítulo en un nuevo archivo HTML vacío. A continuación, abra el archivo en su navegador para ponerlos a prueba.

HTML5 permite el uso de Javascript dentro de los elementos HTML, pero por las mismas razones que estudiamos con CSS, esta práctica no es recomendable: hace que el código HTML se extienda innecesariamente y que sea difícil de mantener y actualizar, lo que complica el desarrollo de aplicaciones útiles.

4.3.2 Incrustado

Para trabajar con códigos extensos y funciones personalizadas podemos agrupar el código con las etiquetas <script>. El elemento <script> actúa exactamente igual que el elemento <style> de CSS: organiza el código en un solo lugar y se refiere al resto de los elementos del documento utilizando referencias. No hace falta utilizar el atributo type para especificar el lenguaje en la etiqueta <script> pues en HTML5 se asigna Javascript por defecto.

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Javascript</title>
<script>
  function hazclic(){
    var elem = document.getElementsByTagName('p')[0];
    elem.addEventListener('click', mostrarAlerta);
  }
  function mostrarAlerta(){
    alert('¡Has hecho clic!');
  }
  addEventListener('load', hazclic);
</script>
</head>
<body>
<section id="Principal">
  <p>Pulsa aquí</p>
  <p>Aquí no puedes pulsar</p>
</section>
</body>
</html>
```

Código 4-37

Javascript incrustado.

El elemento <script> y su contenido puede ser ubicado en cualquier parte del documento, dentro de otros elementos o entre ellos. Para una mayor claridad, se recomienda localizar siempre el código Javascript dentro del encabezado del documento (como en el ejemplo del Código 4-37) y luego usar referencias para acceder a los elementos HTML desde el código Javascript. El objeto Document incluye tres métodos para este propósito:

getElementsByTagName(): Hace referencia los elementos por palabra clave o nombre.

getElementById(): Se refiere a los elementos por el valor de su atributo id.

getElementsByClassNombre(): Es un nuevo método que permite hacer referencia a los elementos utilizando el valor de su atributo class.

Aun si sigue la práctica recomendada de colocar los scripts dentro de la cabeza del documento, debe tener en cuenta que el código es leído secuencialmente por el navegador y por tanto no se puede hacer referencia a un elemento antes de que haya sido creado.

En el Código 4-37, la secuencia de comandos se coloca en la cabecera del

documento y por tanto se lee antes de la creación de los elementos <p>. Si se hubiera intentado modificar el elemento <p> de este código con una referencia, se habría recibido un mensaje de error explicando que el elemento no existe. Para evitar este problema, la referencia al elemento y el registro del evento se incluyen en una función llamada hazclic(), y el código que responde a ese evento se inserta en una segunda función llamada mostrarAlerta().

Al usar esta técnica para agrupar el código Javascript, no solo hay que hacer referencia a los elementos, sino que también hay que registrar controladores de eventos para esos elementos. Con este fin, los atributos de eventos son sustituidos por el método addEventListener(), que añade detectores del evento click para el elemento y del evento de load para la ventana.

Vamos a echar un vistazo a la ejecución de la totalidad del documento del Código 4-37. En primer lugar, las funciones se declaran pero no se ejecutan, a continuación se crean los elementos HTML, incluidos los elementos <p> y finalmente, cuando se ha cargado todo el documento, se ejecuta el evento load de la ventana y luego se llama a la función hazclic(). En esta función, el método getElementsByTagName hace referencia a los elementos <p>. Este método pertenece al objeto del documento, por lo que tenemos que utilizar la propiedad document para llamarlo. Devuelve una matriz que contiene una lista de los elementos que se encuentran en el documento. Sin embargo, utilizando el índice [0] al final del método, se indica que solo debe ser seleccionado el primer elemento. Una vez que se identifica este elemento, se registra un controlador de eventos para el evento clic mediante la propiedad de evento onclick. Se define como controlador a la función mostraralerta(), que se ejecutará cada vez que se dispare el evento, mostrando una pequeña ventana con el mensaje ¡Has hecho clic!.

Puede parecer un montón de código y esfuerzo para reproducir el mismo efecto logrado con un solo atributo en el ejemplo del Código 4-36, pero teniendo en cuenta el potencial de HTML5 y la complejidad alcanzada por las aplicaciones de Javascript, la concentración de código en un lugar y una adecuada

organización representa una gran ventaja para futuras implementaciones y para crear sitios web y aplicaciones más fáciles de desarrollar y mantener.

Conceptos básicos

En la última línea del Código 4-37, no usamos paréntesis para llamar a la función hazclic(), ya que pasamos la referencia de la función a la propiedad de evento y no el resultado de la ejecución de la función.

4.3.3 Desde un archivo externo

Los códigos Javascript crecen exponencialmente con la adición de nuevas funciones y la aplicación de algunas de las API que estudiaremos más tarde. Los códigos incrustados hacen que nuestros documentos sean más grandes y repetitivos. Para reducir el tiempo de descarga de nuestros documentos, aumentar nuestra productividad, y distribuir y reutilizar nuestros códigos en varios los documentos sin comprometer su eficiencia, se recomienda guardar los códigos Javascript en uno o más archivos externos, y posteriormente llamarlos mediante el atributo src.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Javascript</title>
  <script src="micodigo.js"></script>
</head>
<body>
  <section id="principal">
    <p>Pulsa aquí</p>

    <p>No puedes pulsar aquí</p>
  </section>
</body>
</html>
```

Código 4-38

Presentación de código Javascript desde un archivo externo.

El elemento de <script> del Código 4-38 carga el código Javascript desde un archivo externo llamado micodigo.js. A partir de ahora será posible insertar el archivo en todos los documentos del mismo sitio web y reutilizar el código en cualquier momento que se desee. Desde la perspectiva del usuario, esta práctica reduce el tiempo necesario para descargar y acceder a la página web, desde la perspectiva del programador, que simplifica el desarrollo y mantenimiento.

Hágalo usted mismo

Copie el Código 4-38 en el archivo HTML creado anteriormente. Cree un nuevo archivo vacío llamado micodigo.js y copie las instrucciones Javascript del Código 4-37. Tenga en cuenta que solo tiene que copiar el código que está entre las etiquetas <script>, sin incluir las etiquetas.

Vale la pena mencionar que hay dos atributos más disponibles para el elemento <script>: se trata de async y defer. Estos son atributos booleanos que indican cómo y cuando se ejecutará el script incluido en el atributo src. Si se declara async, el script se ejecuta sin sincronizar. Si se declara el atributo defer, el script se ejecuta cuando el documento ha sido analizado. Y cuando no se declara ninguno de los argumentos, el script se ejecuta inmediatamente (como ocurrió en nuestro ejemplo).

4.4 Nuevos selectores

Los elementos HTML tienen que ser referidos por Javascript para poder ser modificados por el código. CSS, y más aún en su versión CSS3, tiene un potente sistema de referencia y selección que supera con creces los pocos métodos actualmente proporcionados por Javascript. Los métodos getElementById, getElementsByTagName y getElementsByClassNombre son insuficientes para la integración que este lenguaje necesita y la relevancia que tiene en la especificación HTML5. Para llevar Javascript al nivel requerido, han sido incorporados mejores alternativas. Ahora es posible seleccionar elementos HTML que aplicando todo tipo de selectores CSS utilizando los nuevos métodos querySelector(), querySelectorAll() y matchesSelector().

4.4.1 querySelector()

El método querySelector() devuelve el primer elemento que coincide con el grupo de selectores CSS indicado entre paréntesis. Los selectores se declaran usando comillas y sintaxis CSS, como en el Código 4-39.

```
function hazclic(){
  var elem = document.querySelector("#principal p:first-child");
  elem.addEventListener('clic', mostraralerta);
}
function mostraralerta(){
  alert('¡Has hecho clic!');
}
addEventListener('load', hazclic);
```

Código 4-39

Uso de querySelector().

En el Código 4-39, el método getElementsByTagName que usamos en el script del Código 4-37 ha sido sustituido por querySelector(). Los selectores para esta consulta en particular hacen referencia al primer elemento <p>, que es un elemento secundario del elemento identificado por el atributo id con el valor principal. Como este método solo devuelve el primer elemento encontrado, la pseudo-clase first-child es redundante. El método querySelector() de nuestro ejemplo devuelve el primer elemento <p> dentro de que es, por supuesto, su primer elemento hijo. El propósito de este ejemplo es mostrar que querySelector() acepta todos los tipos de selectores CSS válidos, y ahora, así como CSS, Javascript también proporciona herramientas importantes para hacer referencia a cualquier elemento en el documento. También puede declarar varios grupos de selectores separados por una coma. El método querySelector() devuelve el primer elemento que coincide con cualquiera de ellos.

Hágalo usted mismo

Copie el Código 4-39 en el archivo micodigo.js y abra el documento que contiene el Código 4-38 en su navegador para ver el método querySelector() en la acción. Repita el procedimiento para los ejemplos que vienen.

4.4.2 querySelectorAll()

En lugar de un elemento, querySelectorAll() devuelve todos los elementos que coinciden con los selectores especificados dentro de los paréntesis. El valor devuelto es una matriz que contiene todos los elementos encontrados en el orden que tienen en el documento (similar al efecto producido por el método getElementsByTagName examinado anteriormente, pero en este caso se comparan selectores CSS).

```
function hazclic(){
  var lista = document.querySelectorAll("#principal p");
  lista[0].addEventListener('click', mostraralerta);
}
function mostraralerta(){
  alert('¡Has hecho clic!');
}
addEventListener('load', hazclic);
```

Código 4-40

Uso de querySelectorAll().

El grupo de selectores proporcionados en el método querySelectorAll() en el Código 4-40 encontrará todos los elementos <p> del documento HTML del Código 4-38 que se encuentran dentro del elemento <section>. Después de la ejecución de la primera línea, la matriz lista tiene dos valores: una referencia al primer elemento de <p> y una referencia al segundo elemento <p>. Debido a que las palabras clave de cada matriz creada empiezan automáticamente desde 0, en la línea siguiente se recupera el primer elemento indicando entre corchetes el valor 0 (lista [0]). Tenga en cuenta que este ejemplo no muestra el potencial de querySelectorAll(). Por lo general, este método se puede usar para modificar varios elementos y uno no solo, como en este caso. Puede utilizar un bucle for para recorrer la lista de los elementos devueltos por el método.

```
function hazclic(){
  var lista = document.querySelectorAll("#principal p");
  for(var f = 0; f < lista.length; f++){
    lista[f].addEventListener('click', mostraralerta);
  }
}
function mostraralerta(){
  alert('¡Has hecho clic!');
}
addEventListener('load', hazclic);
```

Código 4-41

Cómo trabajar con todos los elementos encontrados por querySelectorAll().

En el Código 4-41, en lugar de seleccionar solo el primer elemento, se añade un detector para el evento click para cada evento usando un bucle for. Ahora, todos los elementos <p> dentro del elemento <section> despliegan una pequeña ventana cuando el usuario hace clic en ellos.

Conceptos básicos

La propiedad length está disponible para diversos objetos y establece o devuelve su longitud. En el caso de matrices, se puede utilizar para obtener el número de elementos disponibles en la matriz. En el ejemplo del Código 4-41 usamos esta propiedad para obtener el número de elementos que se encuentran utilizando querySelectorAll() y establecer de este modo la condición para el bucle (se ejecutará mientras f < list.length).

El método querySelectorAll(), igual que querySelector(), puede contener uno o más grupos de selectores separados por una coma. Estos métodos pueden combinarse con otros para alcanzar a los elementos que deseamos. Por ejemplo, en el Código 4-42, se logra el mismo resultado de la secuencia de comandos del Código 4-41 utilizando querySelectorAll() y getElementById() juntos.

```
function hazclic(){
  var lista = document.getElementById("principal").
 querySelectorAll("p");
  lista[0].addEventListener('click', mostraralerta);
}
function mostraralerta(){
  alert('¡Has hecho clic!');
}
addEventListener('load', hazclic);
```

Código 4-42

Combinar métodos.

Usando esta técnica, podemos ver cuán precisos pueden ser estos métodos. Podemos combinar métodos en la misma línea o seleccionar un grupo de elementos y luego realizar una segunda selección con otro método para llegar a elementos específicos dentro del primer grupo.

4.4.3 matchesSelector()

El método matchesSelector() no encuentra elementos, sino que comprueba si los elementos que hemos encontrado son exactamente los que estábamos buscando. También se puede utilizar para determinar si el elemento será devuelto por el selector utilizado por tanto permite confirmar la efectividad de los métodos usados. Es posible combinar métodos en la misma línea o seleccionar un grupo de elementos y luego realizar una segunda selección con otro método para llegar a elementos dentro del primer grupo. Este método usa los mismos selectores CSS que los métodos anteriores, pero solo devuelve los valores booleanos true o false.

```
function hazclic(){
  var elem = document.getElementById('principal');
  var elemp;
  if(elem.webkitMatchesSelector("section")){
    elemp = elem.querySelector("p:first-child")
    elemp.addEventListener('click', mostraralerta);
  }
}
function mostraralerta(){
  alert('¡Has hecho clic!');
}
addEventListener('load', hazclic);
```

Código 4-43

Controlar el elemento encontrado con matchesSelector().

En el script del Código 4-43, el método se utiliza para determinar si el elemento devuelto por el método getElementById() es un elemento de <section> o no. Si es así, entonces el método querySelector() se utiliza para obtener una referencia al primer elemento <p>, y se añade un detector para el evento click.

Importante

El método matchesSelector() es aún experimental y exige el uso de métodos prefijados que ofrecen los proveedores de navegadores, por ejemplo, webkitmatchesSelector() para navegadores basados en WebKit como Google Chrome, y mozmatchesSelector() para Mozilla Firefox.

4.5 Interactuar con el documento

Javascript permite mucho más que hacer referencia a elementos y registrar eventos. También se permite asignar estilos, manipular las clases, obtener información de los elementos, cambiar sus atributos y contenidos, crear nuevos elementos o incluso borrarlos.

4.5.1 Estilos Javascript

De forma similar a lo que pasaba con CSS, hay una lista de estilos que pueden ser asignados a un elemento. Estos estilos son proporcionados a través de las propiedades del objeto style. Debido a que este objeto es accesible desde cualquier elemento del documento, para aplicar un nuevo estilo hay que obtener una referencia al elemento y utilizar sus propiedades.

```
function newstyle(){

var elem = document.getElementById('principal');

elem.style.background = "#990000";

}

addEventListener('load', newstyle);
```

Código 4-44

Asignación de estilos con Javascript

Los nombres de estilos de Javascript son distintos a los de CSS. Como no hubo consenso sobre esta cuestión, es necesario aprenderse el nombre de cada uno de ellos a pesar de que son capaces de proporcionar los mismos valores para las propiedades. Aquí está una lista de los más comunes:

Hágalo usted mismo

Estas son propiedades estándar disponibles desde las primeras versiones de Javascript. Para obtener una lista completa, por favor visite nuestro sitio web y siga los enlaces de este capítulo.

background: Establece o devuelve todos los estilos de fondo. También se puede trabajar con cada estilo individualmente utilizando las propiedades asociadas: backgroundColor, backgroundImage, backgroundRepeat, backgroundPosition y backgroundAttachment.

border: Establece o devuelve los estilos de todo el borde. Se puede modificar cada borde individualmente usando las propiedades asociadas: borderTop, borderBottom, borderLeft y borderRight.

margin: Establece o devuelve los estilos de margen. También se pueden utilizar las propiedades asociadas marginBottom, marginLeft, marginRight y marginTop.

padding: Establece o devuelve los estilos de relleno. También se pueden utilizar las siguientes propiedades asociadas: paddingBottom, paddingLeft, paddingRight y paddingTop.

width: Establece o devuelve el ancho del elemento. Hay dos propiedades asociadas que nos permiten establecer el ancho máximo y mínimo de un elemento: maxWidth y minWidth.

height: Establece o devuelve la altura del elemento. Hay también dos propiedades asociadas que nos permiten ajustar la altura máxima y mínima: maxHeight y minHeight.

visibility: Determina si un elemento es visible o no. Puede tomar dos valores: visible y hidden. Como el resto de las propiedades, también se puede utilizar para devolver el valor actual.

color: Establece o devuelve el color del texto.

font: Establece o devuelve todos los estilos de fuente. También se pueden definir los estilos de forma indidual utilizando las propiedades asociadas: font-Family, fontSize, fontSizeAdjust, fontStyle, fontVariant y font-Weight.

4.5.2 ClassList

El trabajo con estilos de forma individual no es una práctica habitual en el mundo real. Por lo general, los elementos reciben el estilo de un grupo de propiedades CSS asignadas a través del atributo class. Como explicamos en el Capítulo 2, estas reglas se llaman clases. Las clases se definen en las hojas de estilo CSS de forma permanente, ya que no se puede cambiarse pero sí pueden ser sustituidas. Javascript nos permite cambiar las clases asignadas a un elemento mediante la modificación del valor del atributo class.

```
<html lang="es">
<head>
  <title>Javascript</title>
  <style>
    .miclase {
      background: #DDDD00;
    }
  </style>
  <script>
    var elem;
    function iniciar(){
      elem = document.getElementById('principal');
      elem.addEventListener('click', cambiarclase);
    }
    function cambiarclase(){
      elem.className = "miclase";
    }
    addEventListener('load', iniciar);
  </script>
</head>
<body>
  <section id="principal">Mi contenido</section>
</body>
</html>
```

Código 4-45

Cambiar la clase.

La propiedad className establece o devuelve el valor del atributo class. Como se muestra en el Código 4-45, para declarar una nueva clase para un elemento hay que asignar su nombre a la propiedad className. En este ejemplo, se crea la función inicio() para obtener una referencia y se añade un detector para el evento click del elemento <section>. Cuando el usuario hace clic en este elemento, se llama a la función cambiarclase() y se le asigna la clase CSS miclase al elemento, lo que cambia su color de fondo.

Como se ha mencionado en el Capítulo 2, se pueden asignar múltiples clases a un elemento. Si es el caso, en lugar de la propiedad className es mejor aplicar el nuevo objeto classList. Este objeto proporciona los métodos siguientes:

add(clase): Agrega otra clase para el elemento.

remove(clase): Elimina una clase de elemento.

toggle(clase): Agrega o elimina una clase, dependiendo de la situación actual. Si la clase ya ha sido asignada al elemento, se elimina. Si no, la clase se añade al elemento.

contains(clase): Este método detecta si la clase se le asignó al elemento o no

y, de acuerdo a esto, devuelve los valores true o false.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Javascript</title>
  <style>
    .miclase {
      background: #DDDD00;
    }
  </style>
  <script>
    var elem;
    function inicio(){
      elem = document.getElementById('principal');
      elem.addEventListener('click', cambiarclase);
    }
    function cambiarclase(){
      elem.classList.toggle("miclase");
    }
    addEventListener('load', inicio);
  </script>
</head>
<body>
  <section id="principal">Mi contenido</section>
</body>
</html>
```

Código 4-46

Activar y desactivar clases.

Los métodos add() y remove() son similares a className pero son más adecuados para el trabajo con múltiples clases. El método contains() es similar a una declaración if: útil para comprobar si una determinada clase ha sido asignada a un elemento o no. El menos común es toggle(). Este método comprueba el estado del elemento y añade la clase si ésta no ha sido asignada antes o la remueve si no es así.

El documento del Código 4-46 utiliza básicamente el mismo código fuente que el ejemplo anterior, pero la función cambiarclase() ha sido modificada. La propiedad className fue sustituida por el método toggle(). Ahora, cada vez que el usuario haga clic en el elemento <section>, los estilos de este elemento cambiarán, aplicando o eliminando el color de fondo.

Hágalo usted mismo

Sustituya el método toggle() del Código 4-46 por add() o remove() para agregar o quitar la clase de elemento. Por ejemplo, elem.classList add(“miclase”). Trate de crear un código para trabajar de forma similar a la que usamos con toggle() pero aplicando el método contains(). Puede utilizar una sentencia if para comprobar si la clase ha sido asignada al elemento y agregarla o quitarla según el resultado.

4.5.3 Acceder a los atributos

La propiedad className y los métodos del objeto classList que acabamos de discutir son específicos del atributo class. Javascript proporciona otros métodos para trabajar con atributos, obtener su valor, establecer un nuevo valor o eliminarlo:

getAttribute(nombre): Devuelve el valor del atributo especificado por el argumento nombre.

setAttribute(nombre, valor): Establece un nuevo atributo para un elemento. Los parámetros proporcionan el nombre y el valor del atributo: Si el atributo ya existe, el método solo cambiará su valor.

removeAttribute(nombre): Elimina el atributo especificado por el argumento name.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Javascript</title>
  <script>
    var elem;
    function inicio(){
      elem = document.getElementById('principal');
      alert(elem.getAttribute("id"));
    }
    addEventListener('load', inicio);
  </script>
</head>
<body>
  <section id="principal">Mi contenido</section>
</body>
</html>
```

Código 4-47

Obtener el valor de un atributo.

La secuencia de comandos del Código 4-47 utiliza el método getAttribute() para mostrar una ventana emergente con el valor del atributo id del elemento <section>. Tan pronto como se carga el documento, se llama la función inicio() y el método alert() muestra el mensaje en la pantalla.

4.5.4 dataset

Hasta ahora hemos estado trabajando con atributos estándar. En el Capítulo 1 explicamos cómo personalizar elementos con el atributo de data-*. Ésta es una técnica importante para añadir información a un elemento. Recordará que la información no es visible para el usuario, pero sí es accesible para el documento y el navegador. Para leer este tipo de atributos, Javascript proporciona la propiedad dataset.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Javascript</title>
  <script>
    function inicio(){
      var elem = document.getElementById('principal');
      alert(elem.dataset.total);
    }
    addEventListener('load', inicio);
  </script>
</head>
<body>
  <section id="principal" data-total="3">Mi contenido</section>
</body>
</html>
```

Código 4-48

Lectura de atributos personalizados.

El atributo de data-* Puede tomar cualquier valor en lugar del símbolo *, y permite añadir todos los atributos que sea necesario para un solo elemento. El valor elegido para el nombre de cada atributo será el que permitirá identificarlos más tarde usando dataset. Esta propiedad devuelve un objeto con todos los atributos data-* definidos en el elemento. Para obtener el valor de un atributo específico, solo tiene que utilizar su nombre como una propiedad de dataset.

En el Código 4-48, se muestra en la pantalla el valor del atributo personalizado de data-total tan pronto como se carga el documento, gracias a la instrucción alert (elem.dataset.total)).

4.5.5 Crear y borrar elementos

Igual que hicimos con los atributos, podemos crear o borrar elementos HTML desde Javascript. Básicamente, se utilizan los siguientes tres métodos para este propósito:

createElement(palabra clave): Crea un nuevo nodo del elemento del tipo especificado por la palabra clave. No añade el elemento al documento.

appendChild(elemento): Este método se usa junto con createElement() para insertar un nuevo elemento como hijo de un elemento existente en el documento. El atributo especificado debe ser el valor devuelto por el método createElement().

removeChild(elemento): Este método elimina un elemento hijo de otro elemento. El atributo especificado debe ser una referencia al elemento hijo que debe ser eliminado.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Javascript</title>
  <script>
    function inicio(){
      var elem = document.getElementById('principal');
      var elementonuevo = document.createElement('p');
      elem.appendChild(elementonuevo);
    }
    addEventListener('load', inicio);
  </script>
</head>
<body>
  <section id="principal">Mi contenido</section>
</body>
</html>
```

Código 4-49

Adición de nuevos elementos al documento.

El método createElement() pertenece al objeto document. En el Código 4-49, lo usamos para crear un nuevo elemento <p>. Este nuevo elemento se agrega como un elemento secundario del elemento <section> gracias al método appendChild().

El método createElement() agrega un nuevo elemento al DOM (Modelo de objeto del documento), pero no al documento HTML. Como hemos mencionado antes, el DOM es una estructura en árbol de los objetos creada por el navegador para representar el documento HTML. Cuando solo hay que trabajar con un objeto, tal como una imagen o un vídeo que será procesado pero no mostrado en la pantalla, este método es más que suficiente. Sin embargo, cuando el elemento debe ser parte del documento HTML, debe agregarse utilizando appendChild().

4.5.6 innerHTML, outerHtml e insertAdjacentHTML

En el ejemplo anterior se añadió un nuevo elemento <p> al documento pero no se mostró nada en la pantalla porque el elemento estaba vacío. Hay diversas maneras de agregar contenido a un elemento de Javascript. La forma más flexible (y estándar en HTML5) es la propiedad innerHTML. Esta propiedad establece o devuelve el contenido de un elemento.

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Javascript</title>
<script>
  function inicio(){
    var elem = document.getElementById('principal');
    var elementonuevo = document.createElement('p');
    elem.appendChild(elementonuevo);
    elementonuevo.innerHTML = "¡Hola!";
  }
  addEventListener('load', inicio);
</script>
</head>
<body>
<section id="principal">Mi contenido</section>
</body>
</html>
```

Código 4-50

Añadir contenido a un elemento.

Todo lo que necesitamos para añadir nuevos contenidos es una referencia al elemento que se desea modificar. En el Código 4-50, la referencia se toma del valor devuelto por el método createElement(), que es elementonuevo, pero podemos utilizar cualquier referencia devuelta por métodos de referencia como getElementById(), como se verá en ejemplos posteriores.

El valor proporcionado por la propiedad innerHTML puede ser texto o código HTML. Por lo general se recomienda añadir nuevos elementos HTML usando la combinación de createElement() y appendChild () y aplicar innerHTML (o también textContent) cuando se trabaja solo con texto. Esto se debe a que innerHTML es más eficiente para reemplazar el contenido que para añadir nuevo.

En el siguiente ejemplo el contenido actual del elemento <section> se lee y se almacena en la variable content. Entonces, se añade este valor además de un nuevo contenido al elemento. Utilizando esta propiedad es posible añadir elementos y contenido sin aplicar ningún método.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Javascript</title>
  <script>
    function inicio(){
      var elem = document.getElementById('principal');
      var content = elem.innerHTML;
      elem.innerHTML = content + "<p>¡Hola!</p>";
    }
    addEventListener('load', inicio);
  </script>
</head>
<body>
  <section id="principal">Mi contenido</section>
</body>
</html>
```

Código 4-51

Adición de elementos con innerHTML.

Junto con innerHTML, HTML5 incluye una serie de propiedades y métodos para trabajar con el contenido.

textContent: Esta propiedad establece o devuelve el contenido de un elemento como texto.

outerHTML: Esta propiedad establece o devuelve el contenido del documento. A diferencia de innerHTML, esta propiedad sustituye no solo el contenido, sino también el elemento al que se aplica.

insertAdjacentHTML (lugar, contenido): Este método inserta contenido en una localización de acuerdo con el valor del primer atributo. Los valores disponibles son beforebegin, que lo inserta antes de que el elemento; afterbegin, que lo inserta dentro del elemento antes de su primer hijo; beforeend, que lo hace dentro del elemento pero después de su último hijo, y afterend, que lo inserta después del elemento.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Javascript</title>
  <script>
    function inicio(){
      var elem = document.getElementById('principal');
      elem.insertAdjacentHTML("beforebegin", "<section>¡Hola!</
 section>");
    }
    addEventListener('load', inicio);
  </script>
</head>
<body>
  <section id="principal">Mi contenido</section>
</body>
</html>
```

Código 4-52

Agregar una nueva sección antes de <section>.

Además de innerHTML, insertAdjacentHTML() también puede insertar código HTML en el documento. El uso de este método en el Código 4-52 permite añadir un nuevo elemento a <section>.

Importante

Cuando aplicamos innerHTML, el código HTML y los códigos Javascript son interpretados. Por razones de seguridad, usualmente es mejor usar textContent. Deberá tomar una decisión de acuerdo con las exigencias de su aplicación y el nivel de seguridad requerido.

4.6 Las API

Si usted tiene alguna experiencia en programación o ha seguido el capítulo hasta este punto, ya se habrá dado cuenta de la gran cantidad de código que hace falta para realizar sencillas tareas. Imagínese todo el trabajo necesario para construir un sistema de base de datos desde cero, para generar gráficos complejos en la pantalla o para crear una aplicación para la edición de fotografías, por ejemplo.

Javascript es tan poderoso como cualquier otro lenguaje de programación y por las mismas razones que los lenguajes de programación profesional tienen librerías para crear elementos gráficos, motores 3D para videojuegos o interfaces para acceder a bases de datos, entre otras cosas, Javascript tiene sus propias librerías para ayudar a los programadores a lidiar con cuestiones complejas.

4.6.1 API nativas

Las API de Javascript (Interfaces de programación de aplicaciones) son, de hecho, interfaces faciles de utilizar para acceder a librerías complejas. Estas API junto con sus correspondientes librerías, se incluyen de forma nativa en los navegadores compatibles con HTML5 y están listas para ser utilizadas. Proporcionan acceso a los objetos con métodos, propiedades y eventos para sacar el máximo provecho de las librerías disponibles.

Importante

En la actualidad hay docenas de librerías Javascript disponibles. Aunque este no es el tema del presente libro, puede visitar nuestro sitio web y seguir los enlaces de este capítulo para obtener una lista de las librerías más populares en uso.

HTML5 incluye varias APIs para hacer frente a problemas comunes como la generación de gráficos, almacenamiento, comunicación, etc. El potencial de estas interfaces es tan importante que en breve se convertirá en nuestro principal tema de estudio.

4.6.2 API externas

HTML5 fue desarrollado para expandir la Red con un conjunto estándar de tecnologías que pudiera ser soportado por todos los navegadores y proporcionara todo lo que un desarrollador necesita. De hecho, HTML5 fue concebido para ser independiente de cualquier tecnología de terceros. Pero por una razón u otra, a menudo se necesita ayuda adicional.

Antes de la aparición de HTML5, fueron desarrolladas varias librerías Javascript para superar las limitaciones de las tecnologías disponibles. Algunas de estas librerías fueron creadas con fines específicos que iban desde el procesamiento y validación de las formas, hasta la generación y manipulación de gráficos. Estas librerías se han hecho en extremadamente populares y algunas de ellas, como Google Maps, son casi imposibles de imitar por desarrolladores independientes.

Incluso cuando las futuras especificaciones proporcionen mejores métodos o nuevas API nativas, los programadores siempre encontrarán una manera más fácil de solucionar cuestiones complejas. Las librerías que simplifican tareas complejas crecerán y se multiplicarán continuamente y aunque estas librerías no son parte de HTML5, son una parte importante de la web, y algunas de ellas han sido implementadas en las aplicaciones y los sitios web más exitosos de nuestros días. Junto con el resto de las características incorporadas por esta especificación, mejoran Javascript y ayudan a difundir la tecnología de vanguardia.

Una de las librerías externas más populares es jQuery, una librería gratuita que ha sido diseñada para simplificar la creación de aplicaciones modernas con Javascript. Hace que los elementos HTML sean más fácil de seleccionar y las animaciones fáciles de generar. También facilita el control de eventos y ayuda a

implementar Ajax en las aplicaciones.

jQuery tiene la ventaja de proporcionar soporte para navegadores antiguos y hacer que las tareas normales sean más simples y accesibles para cualquier desarrollador. Se puede utilizar junto con HTML5 o un sencillo sustituto de algunas de las características básicas de HTML5 en los navegadores que no están listos para esta tecnología.

Esta librería es simplemente un pequeño archivo que se puede descargar desde www.jquery.com, para después incluirlo en los documentos HTML usando las etiquetas <script>. Proporciona una API sencilla que cualquiera puede entender y aplicar inmediatamente. Una vez que jQuery ha sido incluido en el documento, se pueden utilizar simples métodos agregados por la librería para convertir un sitio web estático en una aplicación moderna y práctica.

4.7 Errores y depuración

Todos los navegadores reportan los errores de Javascript cuando se producen, pero el sistema que utilizan para informar al usuario no ha sido estandarizada. Algunos navegadores despliegan una pequeña ventana que contiene información acerca del error y otros ofrecen consolas para no solo mostrar una lista de errores sino también informar al usuario sobre el procesamiento, mostrar mensajes personalizados y proporcionar herramientas para la depuración.

4.7.1 Consola

La herramienta más sencilla y útil para comprobar si hay errores y depurar es probablemente la consola. Las consolas están disponibles en casi todos los navegadores, pero presentan diferentes formas. Por lo general se componen de varios paneles que detallan información sobre todos los aspectos del documento, incluyendo el código HTML, estilos CSS y, por supuesto, Javascript.

Elements Resources Network Sources Timeline Profiles Audits Console
▶ "XHR Finished Loading: http://www.bbcn.it/services.xml".
▶ "XHR Finished Loading: http://www.bbcn.it/services.xml".

Figura 4-2

Consola de Google Chrome.

La forma de acceder a la consola varía de un navegador a otro, e incluso entre distintas versiones del mismo navegador, como se explica a continuación:

Google Chrome: Este navegador ofrece una consola con información sobre el documento, los códigos HTML, la red, y también el acceso a recursos como el espacio de almacenamiento, bases de datos, caché, etc. La consola Javascript se muestra en la pestaña Console. Para abrir esta consola, pulse la tecla F12 del teclado o vaya al menú principal (un botón en la esquina superior derecha de la ventana) y seleccione Herramientas/Desarrolladores o Herramientas/Cónsola Javascript y active, si hace falta, la pestaña Consola.

Mozilla Firefox: Este navegador ofrece diferentes herramientas para desarrolladores en el submenú Desarrollador web. Aquí encontrará dos opciones para comprobar si hay errores de Javascript: Consola web y Consola de errores. Para acceder a estas herramientas, abra el menú principal (en la versión más reciente deberá usar el botón naranja con el texto Firefox en la esquina superior izquierda de la ventana) y siga alguna de estas rutas: Desarrollador web/Consola web o Desarrollador web/Consola de errores. También hay un buen depurador para Mozilla Firefox llamada Firebug. Para obtener más información y para descargar este complemento, vaya a www.getfirebug.com.

Safari: Este navegador se basa en el mismo motor de Google Chrome y

proporciona una consola similar, pero el acceso a la consola tiene que ser activado por el usuario en el menú Preferencias. Vaya al menú Ver (haga clic en el botón en la esquina superior derecha de la ventana) y seleccione Preferencias. A continuación abra la pestaña Avanzado y marque la casilla con el título Mostrar el menú Desarrollo en la barra de menús. Un nuevo submenú llamado Develop aparecerá en la barra de menús. Seleccione la opción Show Inspector web para abrir la consola.

Opera: Este navegador también ofrece una solución completa de la consola, pero para abrirla es un poco complicado. Puede hacer clic con el botón secundario del ratón sobre la página web y seleccionar la opción Inspeccionar elemento (esta opción también está disponible en otros navegadores) o pulsar las teclas Ctrl.+Mayúsculas+I.

Internet Explorer: El navegador ofrece una consola más sencilla con opciones básicas, pero sigue siendo útil. Está disponible pulsando la tecla F12 o seleccionando la opción Herramientas de desarrollo, en el menú Herramientas.

4.7.2 console.log()

A veces los errores no son errores de programación sino los errores lógicos. El intérprete de Javascript no puede encontrar ningún error en el código, pero la aplicación no está haciendo lo que esperábamos. Puede haber innumerables razones para esto, desde una operación olvidada hasta un conjunto de variables con un valor incorrecto. Para determinar qué es lo que está mal en el código hará falta leer las instrucciones una a una y siguiendo la lógica hasta que algo no tenga sentido. Afortunadamente, está disponible una técnica de programación tradicional llamada puntos de corte o breakpoints.

Importante

En algunos navegadores, el método log() permite el uso de más parámetros entre paréntesis. Del mismo modo, el objeto console incluye otros métodos, además de log(), para fines de depuración. Para obtener más información acerca de este objeto y sus métodos, por favor visite nuestro sitio web y siga los enlaces de este capítulo.

Los puntos de corte detienen la interpretación del código en el punto en el que se encuentran y se insertan en el código para comprobar el estado de la aplicación en el punto indicado. Al llegar a un punto de corte se muestran los valores actuales de las variables o simplemente un mensaje que avisa que el intérprete ha llegado a ese punto.

Tradicionalmente, los programadores de Javascript insertaban un método alert() en algunas partes del código para exponer los valores que podrían proporcionar información de ayuda para encontrar el error, pero este método no resultaba adecuado para muchas situaciones ya que detenía la ejecución del código hasta que la ventana pop fuera cerrada. Para simplificar el proceso y obtener información del código, los navegadores introdujeron el objeto console y el método log(), que muestra el mensaje entre paréntesis en la consola de Javascript.

```
<script>

  var mivariable = 9;

  for(var f = 0; f < mivariable; f++) {

    console.log("El valor actual es " + f);

  }

</script>
```

Código 4-53

Mostrar un mensaje con console.log().

El ejemplo anterior se basa en la secuencia de comandos del Código 4-13. El método alert() ha sido sustituido por console.log() para mostrar un mensaje en cada ciclo del bucle. Estos mensajes solo aparecen en la consola de Javascript y permiten a los desarrolladores comprobar el estado de la aplicación sin interrumpir su ejecución.

Además, console.log() puede imprimir objetos en la consola de Javascript, de forma que los desarrolladores puedan leer el contenido de un objeto desconocido e identificar sus propiedades y métodos.

4.7.3 Evento error

En algunas ocasiones se encontrará con errores que no son culpa suya. A medida que sus códigos se vuelvan más complejos e incorporen sofisticadas API, los errores comenzarán a depender de factores fuera de su control, tales como recursos momentáneamente inaccesibles o un cambio inesperado en el equipo que ejecuta la aplicación. Para ayudar al código a detectar estos errores y corregirlos automáticamente, Javascript proporciona el evento error. Este evento está disponible para varias API, también como un evento general del objeto Window. Aprovechando este evento, podemos programar nuestro código para que sea capaz de responder a nuevas condiciones.

```
<script>
  function mostrarerror(e){
    console.log('Error: ' + e.message);
    console.log('Line: ' + e.lineno);
    console.log('URL: ' + e.filename);
  }
  addEventListener('error', mostrarerror);
  wrongfunction();
</script>
```

Código 4-54

Respuesta a errores de script.

El evento error devuelve un objeto que contienen propiedades con información sobre el error. En el Código 4-54, enviamos mensajes a la consola con los valores de estas propiedades. La propiedad message contiene un texto corto que describe el error, la propiedad lineno proporciona el número de la línea en el código que ha producido el error y la propiedad filename ofrece toda la URL del archivo que contiene el código. En nuestro ejemplo, el error se produce por la ejecución de una función inexistente llamada funcionequivocada(), pero podría ser cualquier error de Javascript. Tenga en cuenta que a veces será mejor utilizar los eventos error específicos de la API, como veremos en capítulos siguientes.

5 Formularios

5.1 Formularios HTML

La Web 2.0 tiene que ver con el usuario. Y cuando el usuario es el centro de atención, lo más relevante es la interfaz y cómo hacerla más intuitiva, natural, práctica y, por supuesto, más atractiva.

Los formularios son las interfaces más importantes de todas, ya que permiten a los usuarios insertar datos, tomar decisiones, comunicar la información y cambiar el comportamiento de una aplicación.

Durante los últimos años se han creado códigos personalizados y librerías para procesar formularios en el equipo del usuario. HTML5 estandarizó estas características, ofreciendo nuevos atributos, elementos y una completa API. Ahora, la capacidad de procesar la información insertada en formularios en tiempo real ha sido incorporada a los navegadores y se ha normalizado completamente.

5.1.1 El elemento <form>

Las formas no han cambiado mucho desde versiones anteriores de HTML. La estructura sigue siendo la misma, pero HTML5 ha añadido nuevos elementos, tipos de entradas y atributos para ampliarlos tanto como sea necesario para proporcionar las características implementadas actualmente en las aplicaciones web.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Formularios</title>
</head>
<body>
  <section>
    <form name="miformulario" method="get" action="file.php">


    </form>
  </section>
</body>
</html>
```

Código 5-1

Uso del elemento <form>.

El elemento <form> define la forma y sus componentes. Tiene etiquetas de apertura y cierre que encierran sus elementos y requiere unos pocos atributos para determinar la forma en la que la información va a ser procesada y enviada:

name: Es el nombre del formulario. Este atributo también está disponible para todos los otros elementos HTML pero es particularmente útil para los elementos de un formulario, como se verá más adelante.

method: Determina el método utilizado para enviar el formulario al servidor. Hay dos valores posibles: get y post. El método get se utiliza para enviar información pública limitada. Los datos se envían a la URL y por lo general no tienen más de 255 caracteres. El método post se utiliza para enviar información privada ilimitada, es decir, datos que no son visibles para el usuario.

action: Declara la URL del archivo en el servidor que procesa la información enviada por el formulario.

target: Determina dónde se mostrará la respuesta enviada por el servidor. Los valores posibles son _blank, para una nueva ventana, _self, para el mismo cuadro, _parent, para el cuadro principal y _top, para la ventana que contiene el marco. El valor _self se establece de forma predeterminada.

Este atributo y sus valores también son aplicables al elemento <a>.

enctype: Declara la codificación que se aplica a los datos enviados por el formulario. Se necesitan tres valores: application/x-www-form-urlencoded (caracteres codificados), multipart/form-data (caracteres no codificados), text/plain (solo espacios codificados). Por defecto se establece el primero.

accept-charset: Este atributo declara el tipo de codificación que se aplica al formulario. Los valores más habituales son UTF-8 e ISO-8859-1. El valor predeterminado es el establecido para el documento en la etiqueta <meta>.

5.1.2. El elemento <input>

El elemento más importante en un formulario es <input>. Este elemento genera un campo de entrada para que el usuario introduzca datos en él. Las características del elemento y el tipo de datos permitidos dependerán del valor del atributo type. Este atributo determina el tipo de entrada que se espera de los usuarios. Hay muchos valores disponibles para el atributo type, pero los más convencionales son los siguientes:

text: Genera un campo de entrada para insertar texto genérico.

hidden: Este valor esconde el campo de entrada. Por lo general se utiliza para enviar información complementaria.

password: Genera un campo de entrada para introducir una contraseña. Enmascara los caracteres para ocultar la información privada. Los caracteres introducidos por el usuario por lo general aparecen en la pantalla como caracteres estrella o como puntos, dependiendo del navegador.

checkbox: Este valor genera una casilla de verificación. Este tipo de entrada requiere la declaración del atributo value para especificar el valor que debe enviarse. El valor se envía solo si la casilla de verificación es seleccionada (si no se especifica un valor, se envía el valor on).

radio: Genera un botón de radio para seleccionar una opción entre varias propuestas. Utilizando el mismo valor para el atributo name, es posible

agrupar varias opciones juntas. El valor enviado será el del atributo value del botón seleccionado. El atributo cheked declara el botón que se selecciona de manera predeterminada.

file: Genera un campo de entrada para seleccionar un archivo del ordenador del usuario.

button: Genera un botón que no realiza ninguna acción, ni envía el formulario por sí mismo. Se trata de un botón de usos múltiples que tiene que ser controlado por el código Javascript (generalmente a través del evento clic. El atributo value se utiliza para declarar el texto que aparecerá en el botón).

submit: Genera un botón para enviar el formulario. Este tipo de entrada usa el atributo value para declarar el texto que aparecerá en el botón. El formulario también se pueden enviar desde Javascript con el uso del método submit(), como se verá más adelante.

reset: Este valor carga una imagen que se utilizará como un botón de envío. El elemento <input> tiene que incluir el atributo src para especificar la dirección URL de la imagen.

El Código 5-2 incluye un ejemplo de los tipos de entrada más comunes. Veremos más ejemplos en los capítulos siguientes:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Formularios</title>
</head>
<body>
  <section id="form">
    <form name="miformulario" method="get" action="file.php">
      <br><label for="minombre">Text: </label>
      <input type="text" name="minombre">
      <br><label for="miopción">Radio: </label>
      <input type="radio" name="opción" value="1" checked> 1
      <input type="radio" name="opción" value="2"> 2
      <input type="radio" name="opción" value="3"> 3
      <br><label for="micontraseña">Password: </label>
      <input type="password" name="micontraseña">
      <br><label for="micasilla">Checkbox: </label>
      <input type="checkbox" name="micasilla" value="123">
      <input type="hidden" value="clave">
      <input type="submit" value="Enviar">
    </form>
  </section>
</body>
</html>
```

Código 5-2

Creación de un formulario con los tipos de entrada más convencionales.

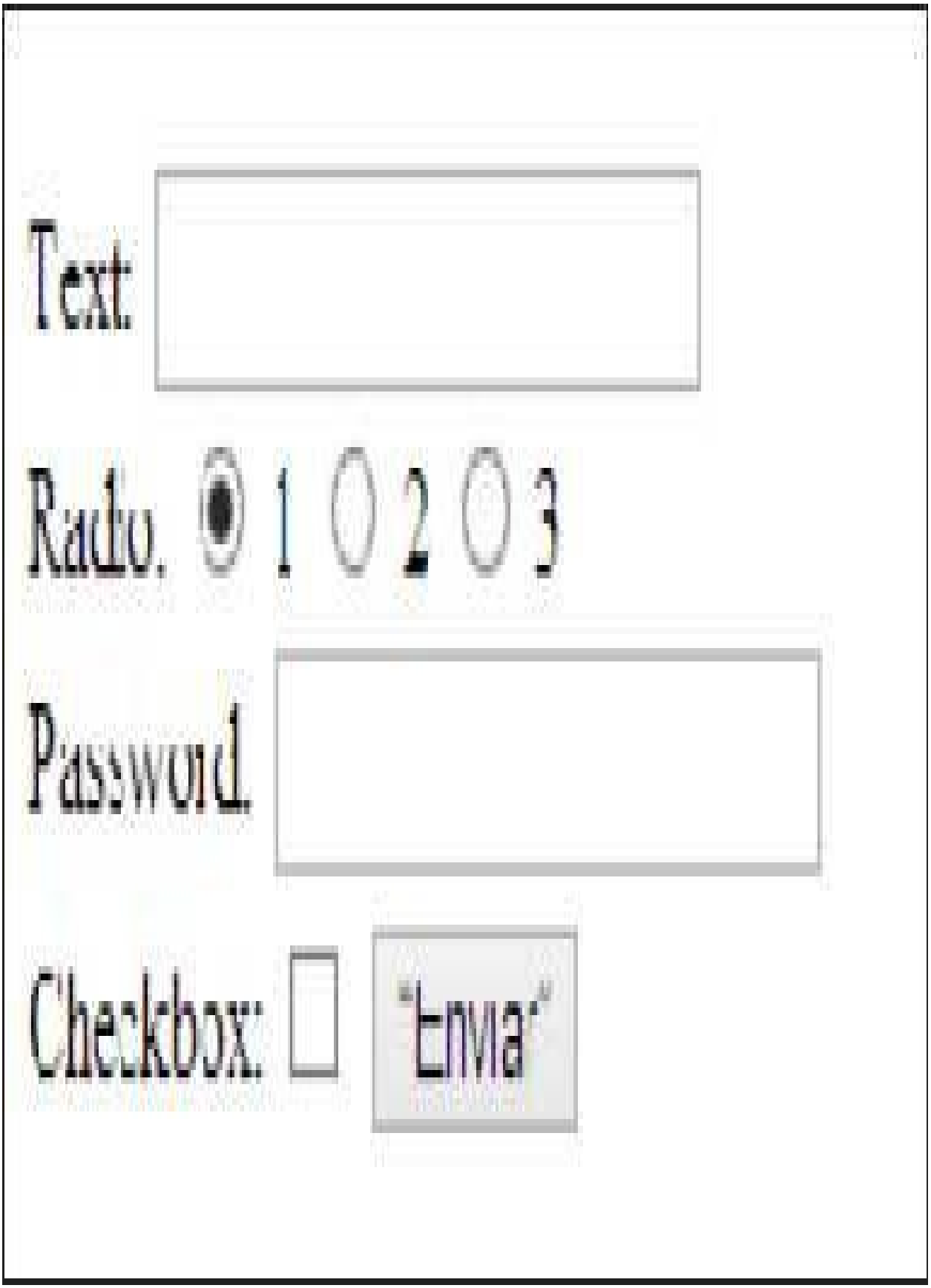

Text:
Radio. 1 0 2 0 3
Password.
Checkbox:
"Envia"

Figura 5-1

Los elementos <input> más convencionales.

Conceptos básicos

El elemento <label> declara la etiqueta para el elemento del formulario. El atributo for de cada elemento se utiliza para asociar la etiqueta con el elemento correspondiente, es decir, el valor de for debe ser el valor del atributo name del elemento al que se refiere la etiqueta.

5.1.3 Más elementos de formulario

Además del elemento <input>, hay otros elementos tradicionales que debe conocer:

<textarea>: Genera un campo de texto de varias líneas. El tamaño puede ser declarado como un número entero utilizando los atributos rows y cols.

<select>: Genera una lista de opciones que se declaran con el elemento <option>.

<option>: Declara cada opción para un elemento <select>.

En el siguiente ejemplo se muestra de forma sencilla la sintaxis de estos elementos. El tamaño del elemento <textarea> es 5 filas y 30 columnas, y se incluyen tres opciones para el elemento <select>.

Los datos enviados por el formulario para éste estarán compuestos por el valor del atributo name del elemento <select> (milista) y el valor del atributo value de la opción seleccionada.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Formularios</title>
</head>
<body>
  <section id="form">
    <form name="miformulario" method="post" action="file.php">
      <label for="mitexto">Textarea: </label>
      <textarea name="mitexto" rows="5" cols="30"></textarea>
      <br><label for="milista">Select: </label>
      <select name="milista">
        <option value="1">Un</option>
        <option value="2">Dos</option>
        <option value="3">Tres</option>
      </select>
      <input type="submit" value="Enviar">
      <input type="reset" value="Restaurar">
    </form>
  </section>
</body>
</html>
```

Código 5-3

Uso de <textarea> y <select>.

Textarea:
Select:
Uno
"Enviado"
Restaurar

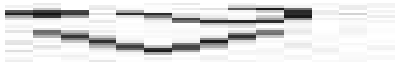
Figura 5-2

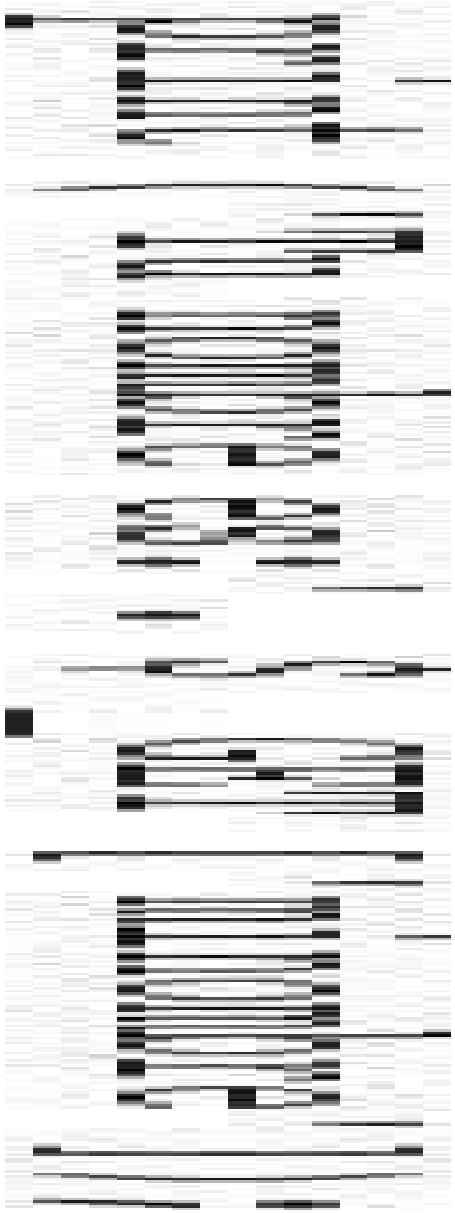
Elementos tradicionales de los formularios.

5.1.4 Enviar un formulario

Cuando se envía un formulario, los datos introducidos se envían utilizando el valor el atributo name de cada elemento y el valor del atributo value correspondiente, que es precisamente el valor introducido por el usuario. Por ejemplo, si insertamos el texto “Roberto” en el primer elemento <input> del formulario del Código 5-2, se enviará minombre = Roberto.

Estas parejas de datos nombre/valor se envían a la URL o se ocultan, de acuerdo con el método indicado por el atributo method del formulario, que como recordará puede ser get o post. Si se establece el método de get, podrá ver los datos en la URL cuando se envía el formulario (por ejemplo, www.minkbooks.com?minombre=Roberto). Para recuperar esta información desde el servidor, tendrá que utilizar una técnica de acuerdo al método establecido para el formulario. Vamos a ver un ejemplo en PHP:





Código 5-4

Procesamiento de datos en el servidor con PHP (file.php).

El código PHP del Código 5-4 muestra un ejemplo de declaraciones que podrían ser parte del archivo file.php declarado como el valor del atributo action en el formulario del Código 5-2. Este archivo es cargado desde el servidor por el navegador cuando se envía el formulario. En el servidor se ejecuta el código PHP y el resultado se devuelve al navegador como un documento HTML. El PHP en nuestro ejemplo utiliza una variable global llamada $_GET para capturar la información enviada al servidor e imprime el valor en el documento que devuelve. Si se cambia el método del formulario por POST, entonces hay que utilizar la variable $_POST en lugar de $_GET.

Importante

Éste es solo un ejemplo de cómo se procesa la información y cómo el navegador transmite al servidor los datos introducidos en un formulario. PHP es un lenguaje que se usa para procesar la información en el servidor, pero no es el único disponible. Para obtener más información al respecto, visite nuestro sitio web y siga los enlaces de este capítulo.

5.2 Nuevos tipos de entrada

HTML5 ha aumentado el número de opciones para el atributo type y ha mejorado el elemento <input>. Ahora los tipos no solo especifican el tipo de entrada que se espera, sino que también le dicen al navegador qué hacer con la información recibida. El navegador procesa los datos de entrada de acuerdo con el valor del atributo type y valida o no la entrada. Este atributo funciona junto con los atributos adicionales para ayudar al navegador a limitar y controlar la entrada del usuario en tiempo real.

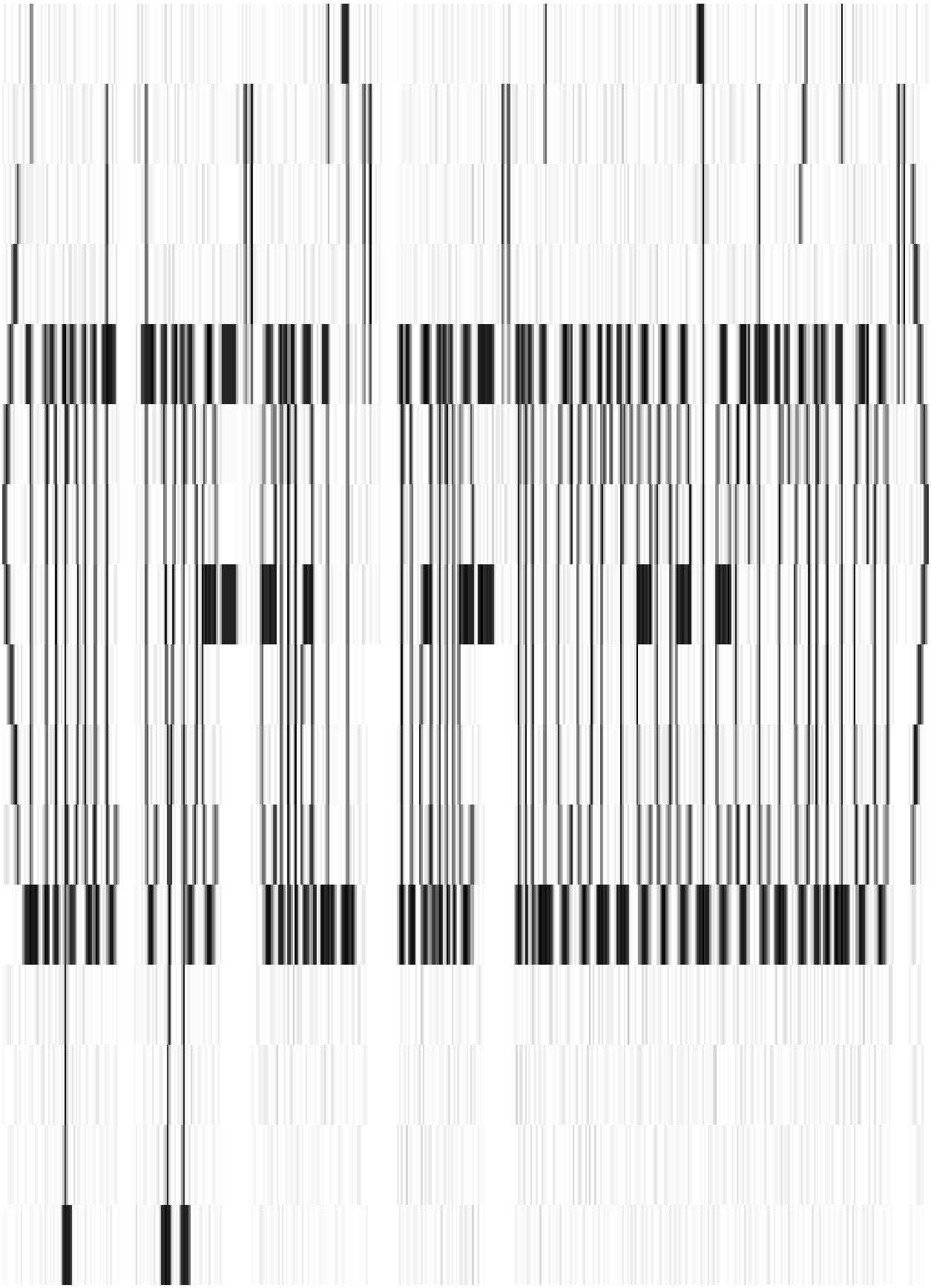
Hágalo usted mismo

Cree un nuevo archivo HTML con el contenido del Código 5-1. Para comprobar cómo funciona cada tipo de entrada, coloque los elementos <input> que desee probar en medio de las etiquetas <form> y abra el archivo en su navegador. En la actualidad, la forma en que estos nuevos tipos de entrada son procesados varía, así que le recomendamos comprobar el código en todos los navegadores disponibles.

5.2.1 Tipo email

Casi todos los formularios actuales tienen un campo de entrada para introducir una dirección de correo electrónico. Pero hasta el momento, el único tipo disponible para este tipo de datos era text.

Como el tipo text representa un texto de carácter general y no datos específicos, había que controlar la entrada con código Javascript para asegurar que el texto escrito era una dirección válida de correo electrónico. Ahora el navegador se encarga de esto gracias a al nuevo tipo email.



Código 5-5

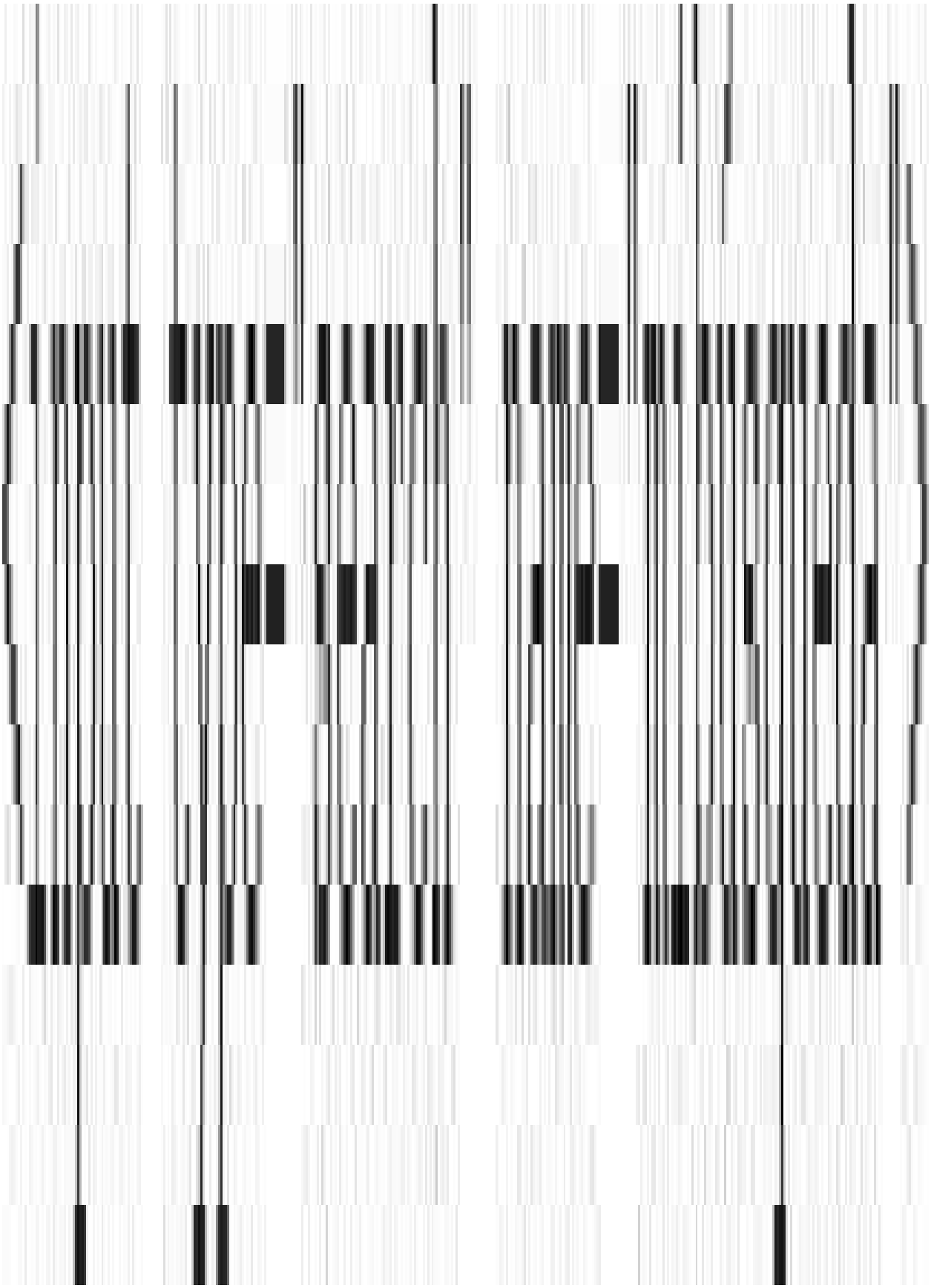
Declaración del tipo email.

El texto introducido en el campo generado por Código 5-5 será comprobado por el navegador y validado como un correo electrónico. Si la validación falla, el formulario no será enviado.

La respuesta del navegador a una entrada inválida no está determinada por la especificación HTML5. Algunos navegadores mostrarán un borde rojo alrededor del elemento <input> que ha producido el error y otros muestran uno azul. Más tarde estudiaremos cómo personalizar este procedimiento.

5.2.2 Tipo search

El tipo llamado search no controla la entrada sino que es solo una indicación de los navegadores. Algunos navegadores cambiarán el diseño por defecto de este elemento para proporcionar al usuario una sugerencia sobre el propósito de este campo.

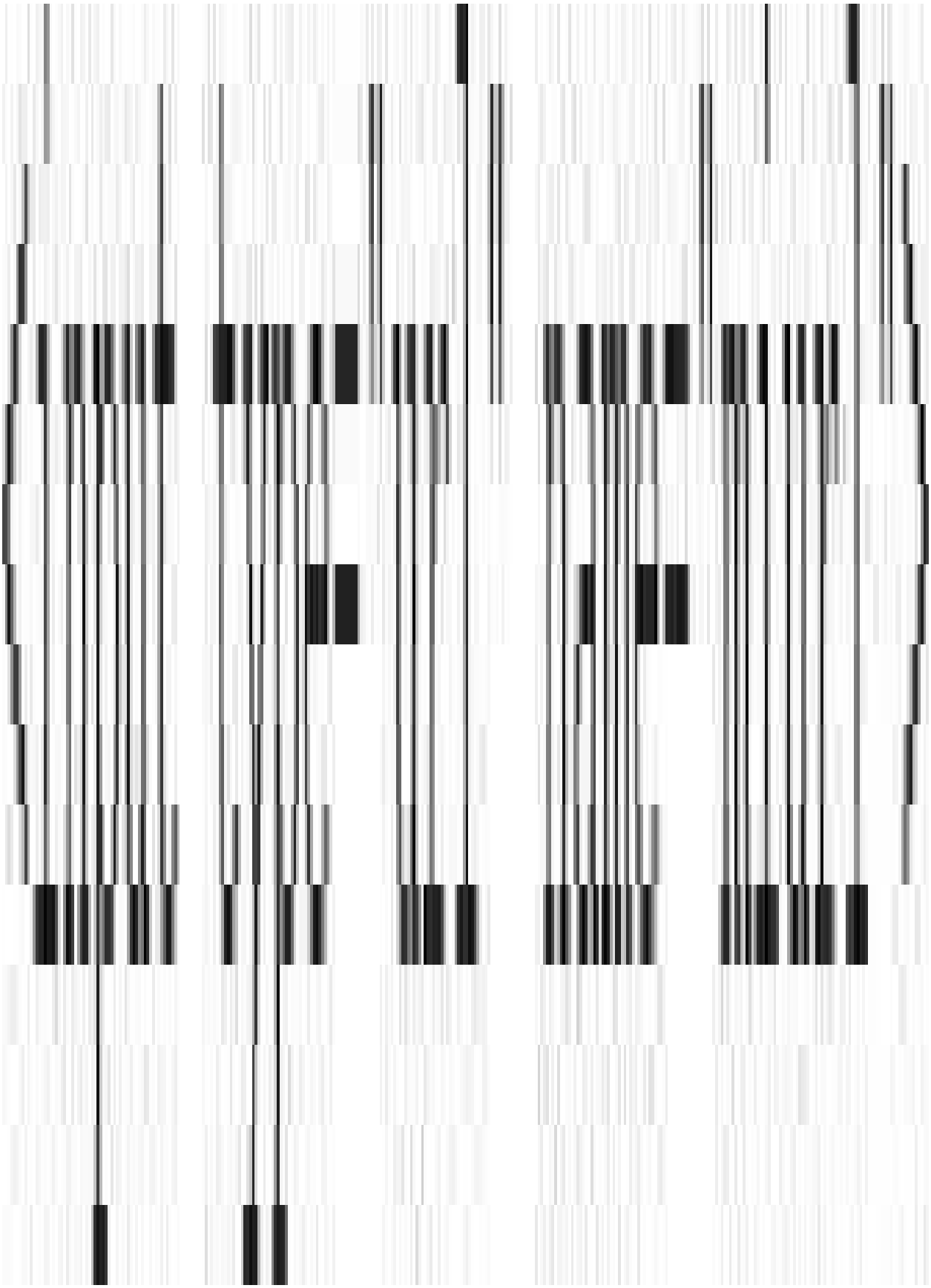


Código 5-6

Declaración del tipo search.

5.2.3 Tipo url

El tipo url funciona exactamente igual que email pero para direcciones Web. Acepta solo direcciones URL absolutas y devuelve un error si el valor es inválido.

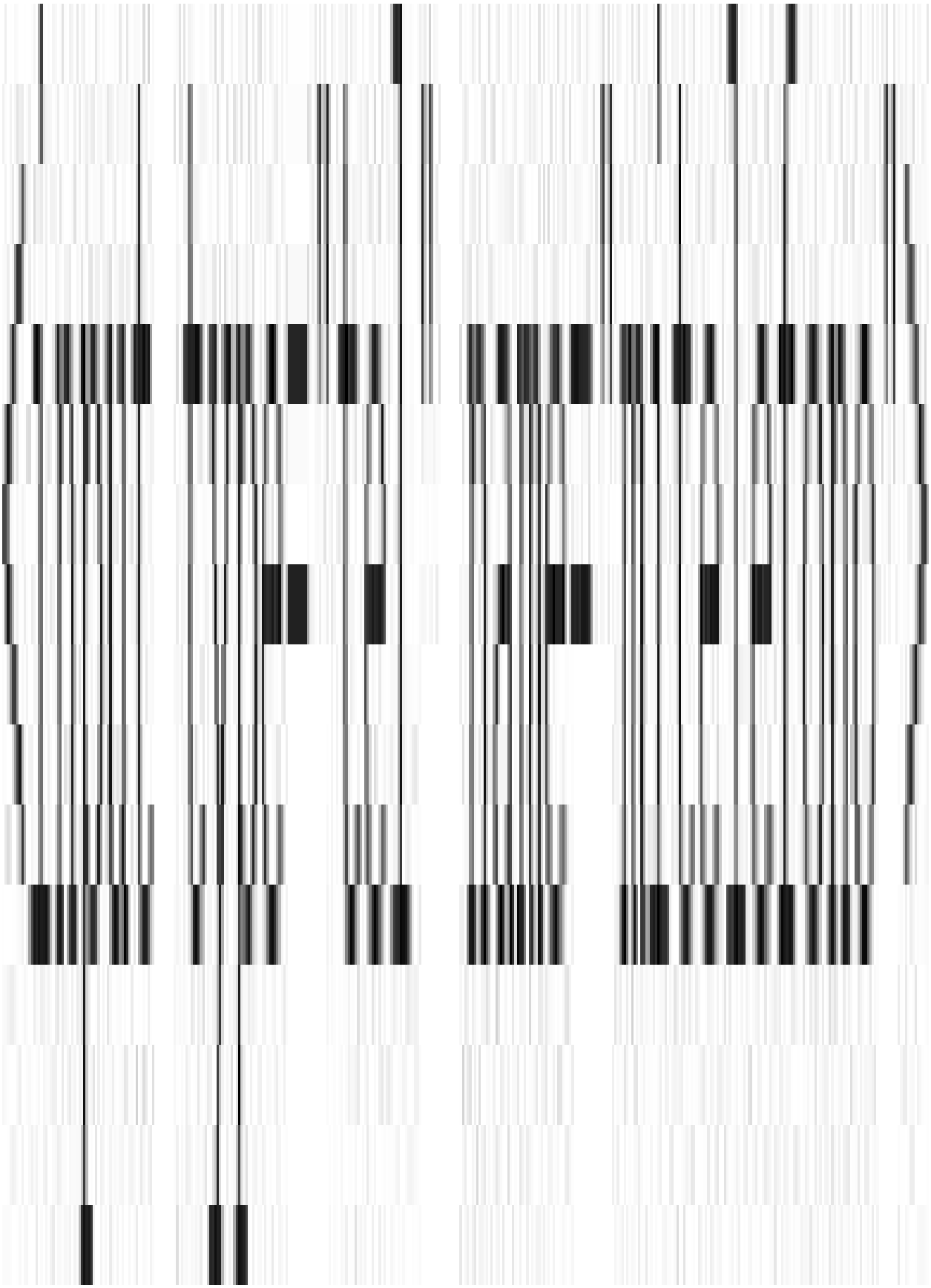


Código 5-7

Declaración del tipo url.

5.2.4 Tipo tel

El tipo tel es para los números de teléfono. A diferencia de los tipos email y url, los tipos tel no obligan a ninguna sintaxis especial. Es una indicación para el navegador en caso de que la aplicación necesite realizar ajustes de acuerdo con el dispositivo en el que está siendo ejecutada.



Código 5-8

Declaración del tipo tel.

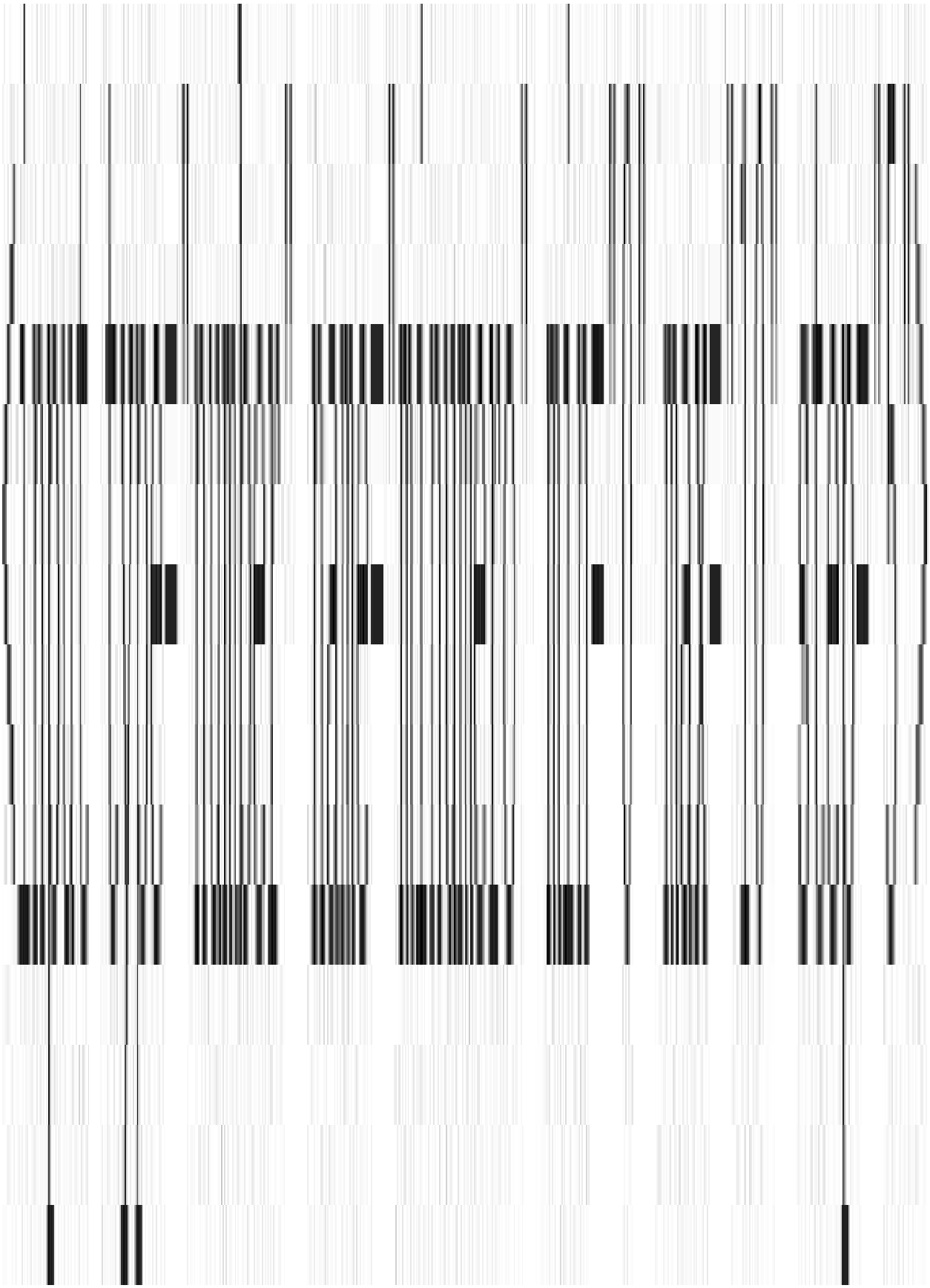
5.2.5 Tipo number

Tal como indica su nombre, el tipo number solo es válido cuando se recibe una entrada numérica. Hay algunos atributos opcionales que pueden ser útiles para limitar este campo:

min: El valor de este atributo determina el valor mínimo aceptable para el campo.

max: El valor de este atributo determina el valor máximo aceptable para el campo.

step: El valor de este atributo determina el valor en el que el campo aumenta o disminuye. Por ejemplo, si se establece un valor de step de 5 con un valor mínimo de 0 y un máximo de 10, el navegador aceptará un valor entre 0 y 5 o entre 5 y 10.



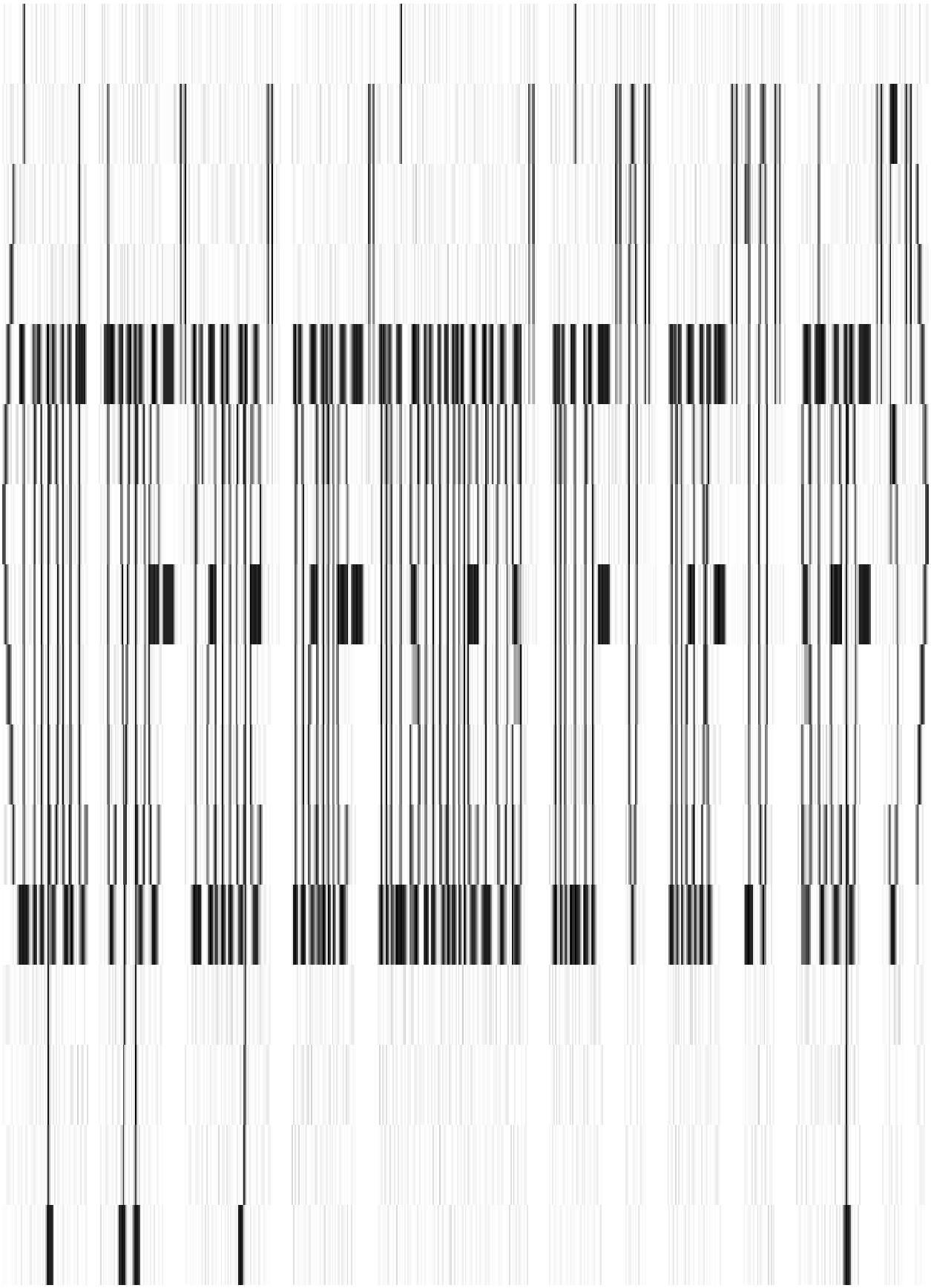
Código 5-9

Declaración del tipo number.

5.2.6 Tipo range

El tipo range hace que el navegador construya un nuevo tipo de controlador que antes no existía. Como su nombre lo indica, este nuevo controlador permite a los usuarios seleccionar un valor de un rango de números. Por lo general se representa con un control deslizante o flechas que mueven el valor hacia arriba y hacia abajo, pero no existe un diseño estándar.

El tipo range utiliza los atributos min y max para establecer los límites del rango. También puede tener el atributo step para declarar el valor en el que el número aumenta o disminuye en cada paso del rango.



Código 5-10

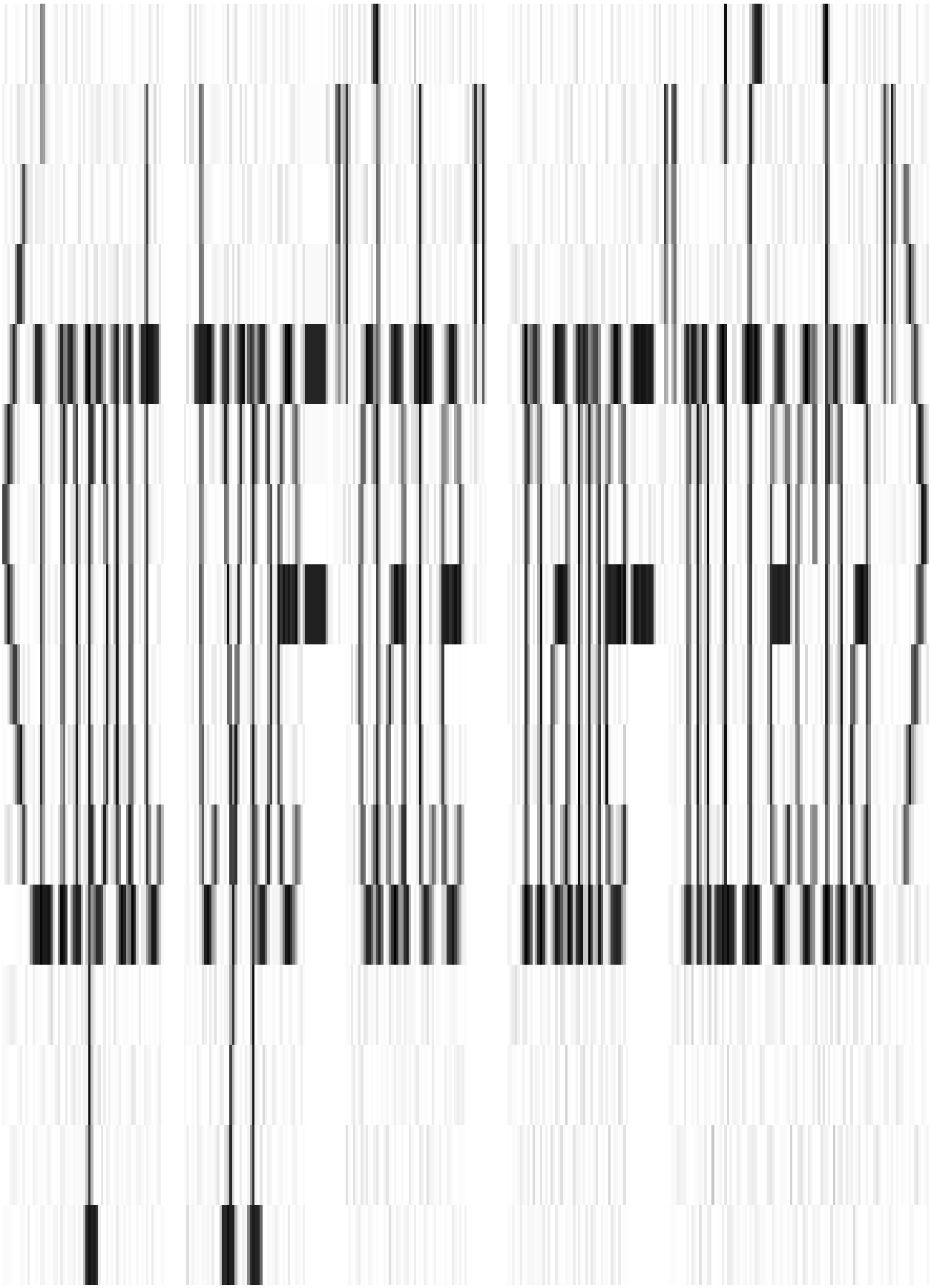
Declaración del tipo range.

Se puede establecer el valor inicial con el atributo value y utilizar Javascript para mostrar los números en pantalla como referencia.

5.2.7 Tipo date

El tipo date genera una nueva clase de controlador: se incluye para proporcionar una mejor forma de introducir la fecha. Los navegadores implementan esta función con un calendario que aparece cada vez que el usuario hace clic en el campo. El calendario permite a los usuarios seleccionar un día que se insertará en el campo de entrada junto con el resto de la fecha. Un ejemplo de esto es el campo utilizado para elegir una fecha para un vuelo.

El navegador construye un calendario con el tipo date, de forma que el programador solo necesite introducir el elemento <input> en el documento para ponerlo a disposición de nuestros usuarios.



Código 5-11

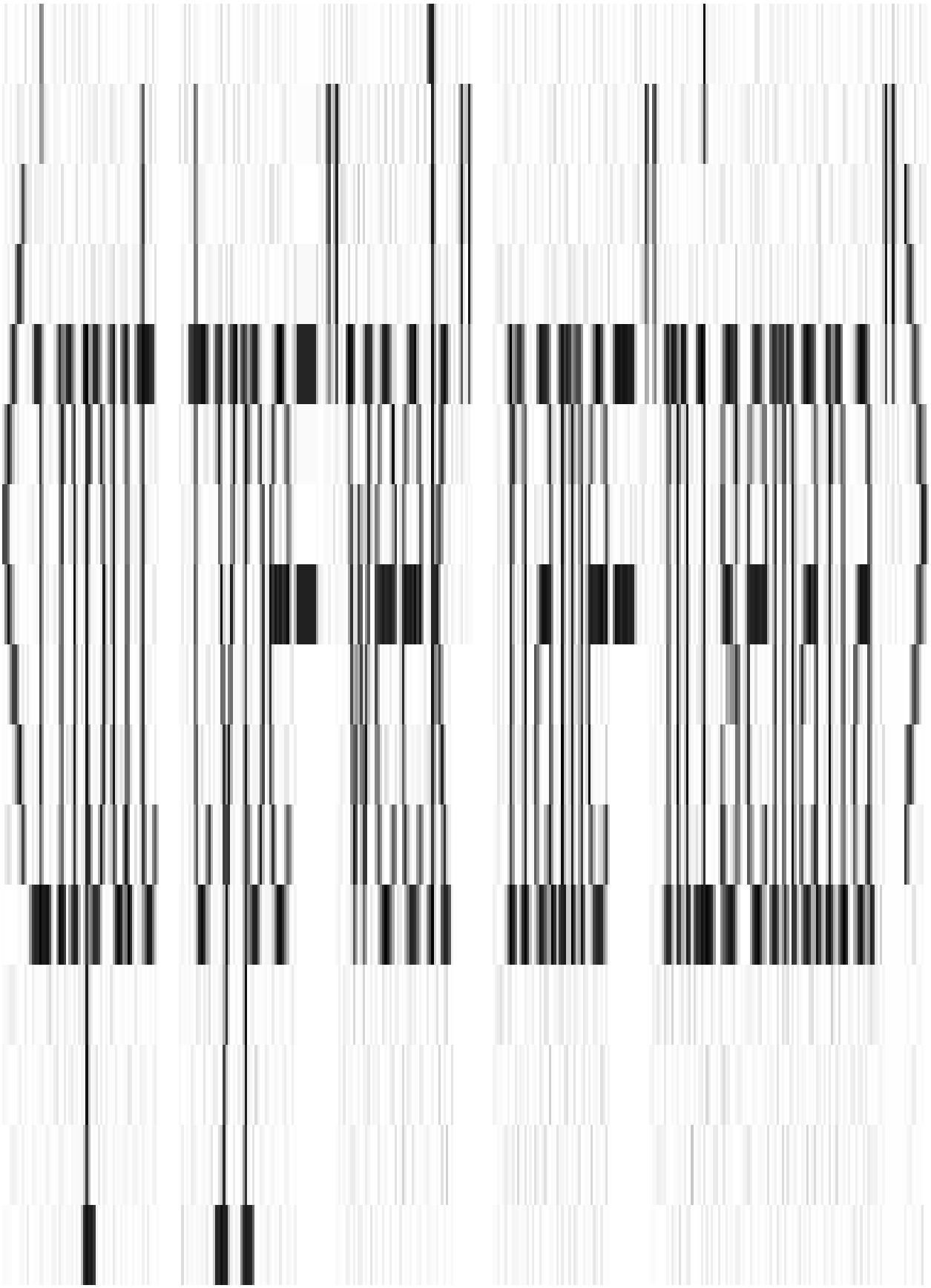
Declaración del tipo date.

La interfaz no es declarada en la especificación. Cada navegador proporciona su propia interfaz y a veces adapta el diseño de acuerdo con el dispositivo en el que se ejecuta la aplicación.

Normalmente el valor generado y esperado tiene la sintaxis siguiente: año-mes-día.

5.2.8 Tipo week

El tipo week proporciona una interfaz de tipo similar a date pero solo recoge semanas enteras. El valor esperado tiene la sintaxis 2012-W50, donde 2012 es el año y 50 es el número de semana.

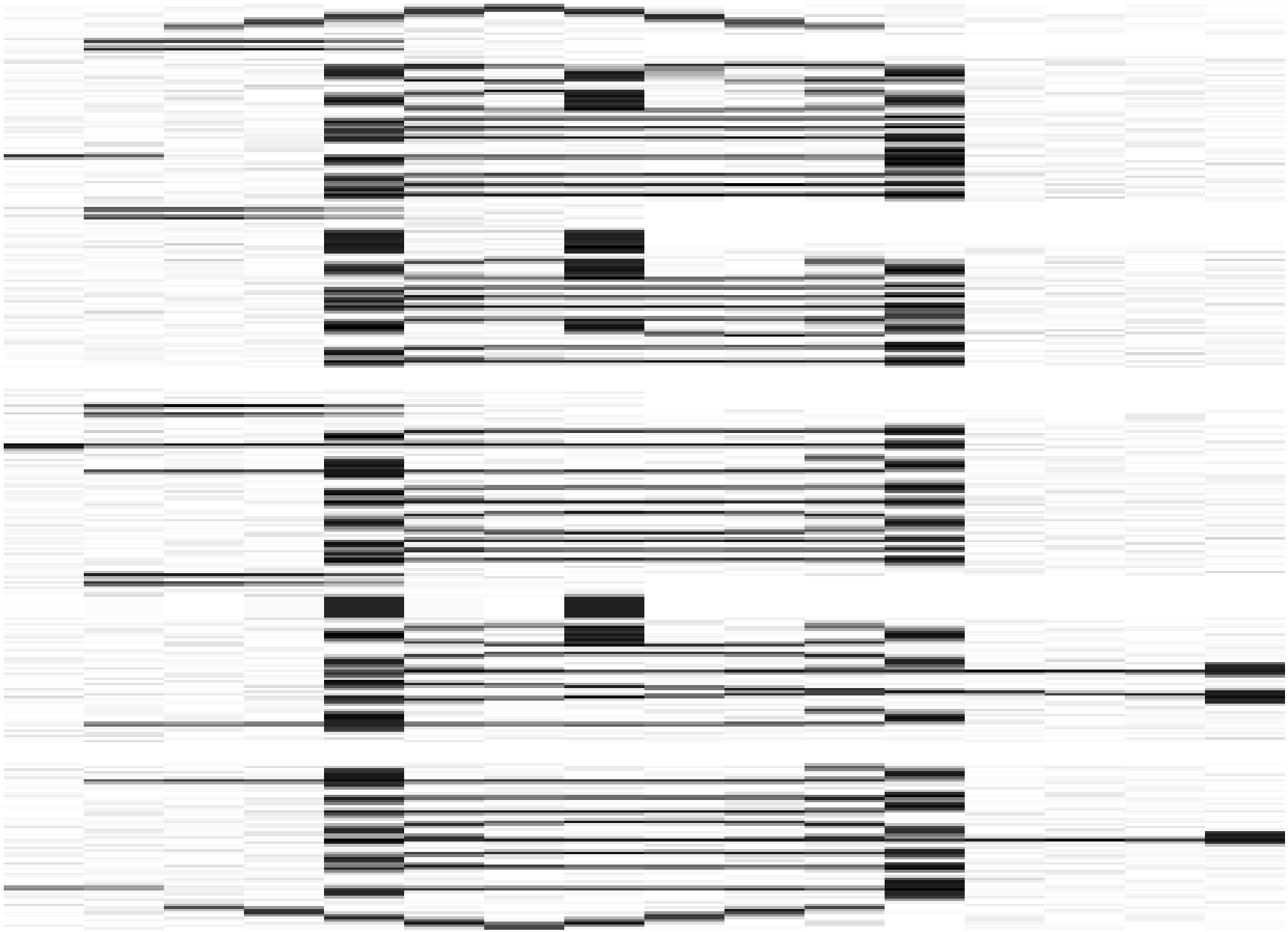


Código 5-12

Declaración del tipo week.

5.2.9 Tipo month

Similar al anterior, el tipo month permite recoger un mes entero. El valor esperado tiene la sintaxis año-mes.

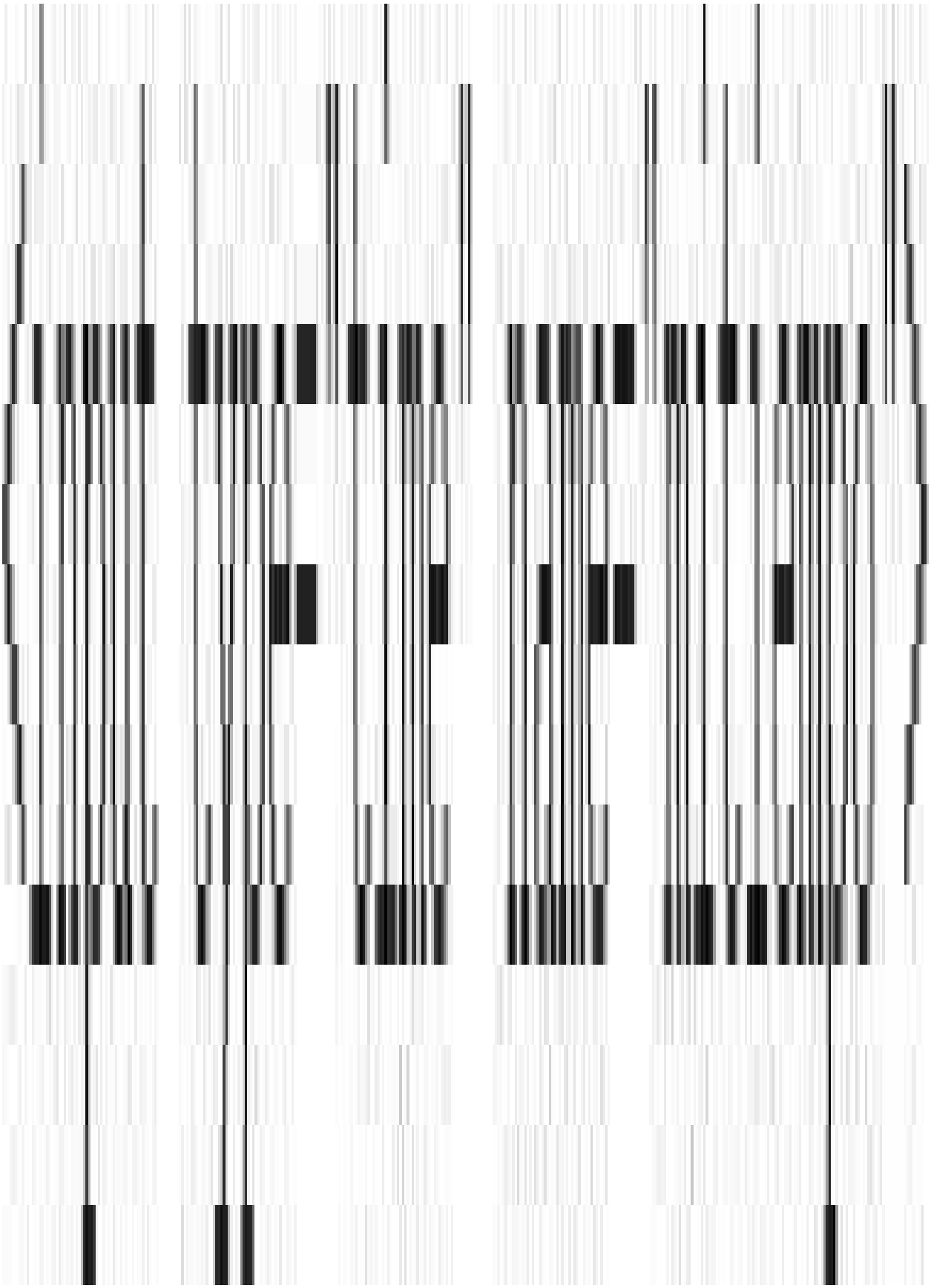


Código 5-13

Declaración del tipo month.

5.2.10 Tipo time

El tipo time es similar a date y se refiere al tiempo. Toma el formato de horas y minutos, aunque en la actualidad su comportamiento también depende de cada navegador. Por lo general, el valor esperado tiene la sintaxis horas:minutos:segudos pero también puede ser horas:minutos.

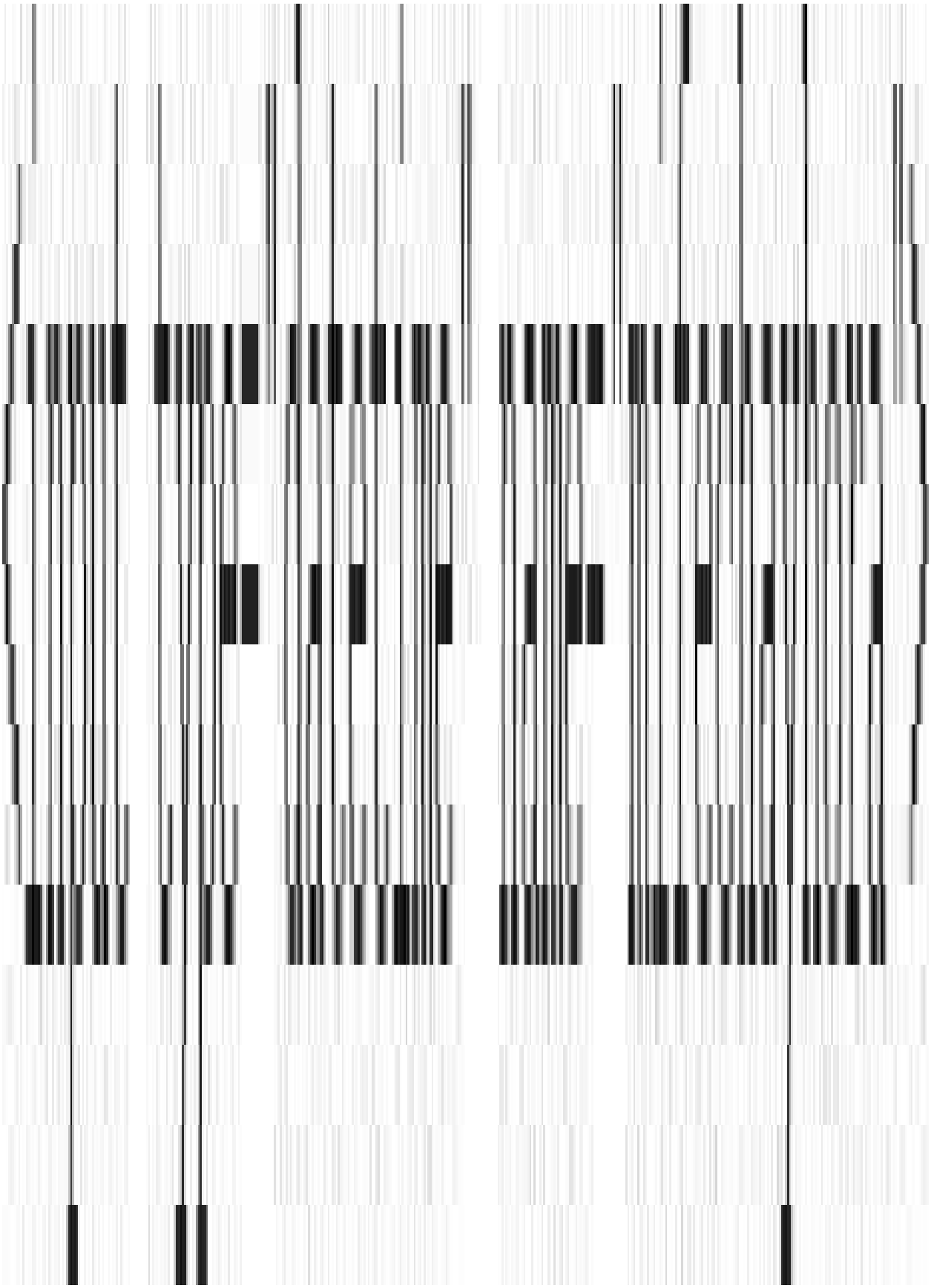


Código 5-14

Declaración del tipo time.

5.2.11 Tipo datetime

El tipo datetime es para la introducción completa de fecha y hora, incluyendo una zona horaria.

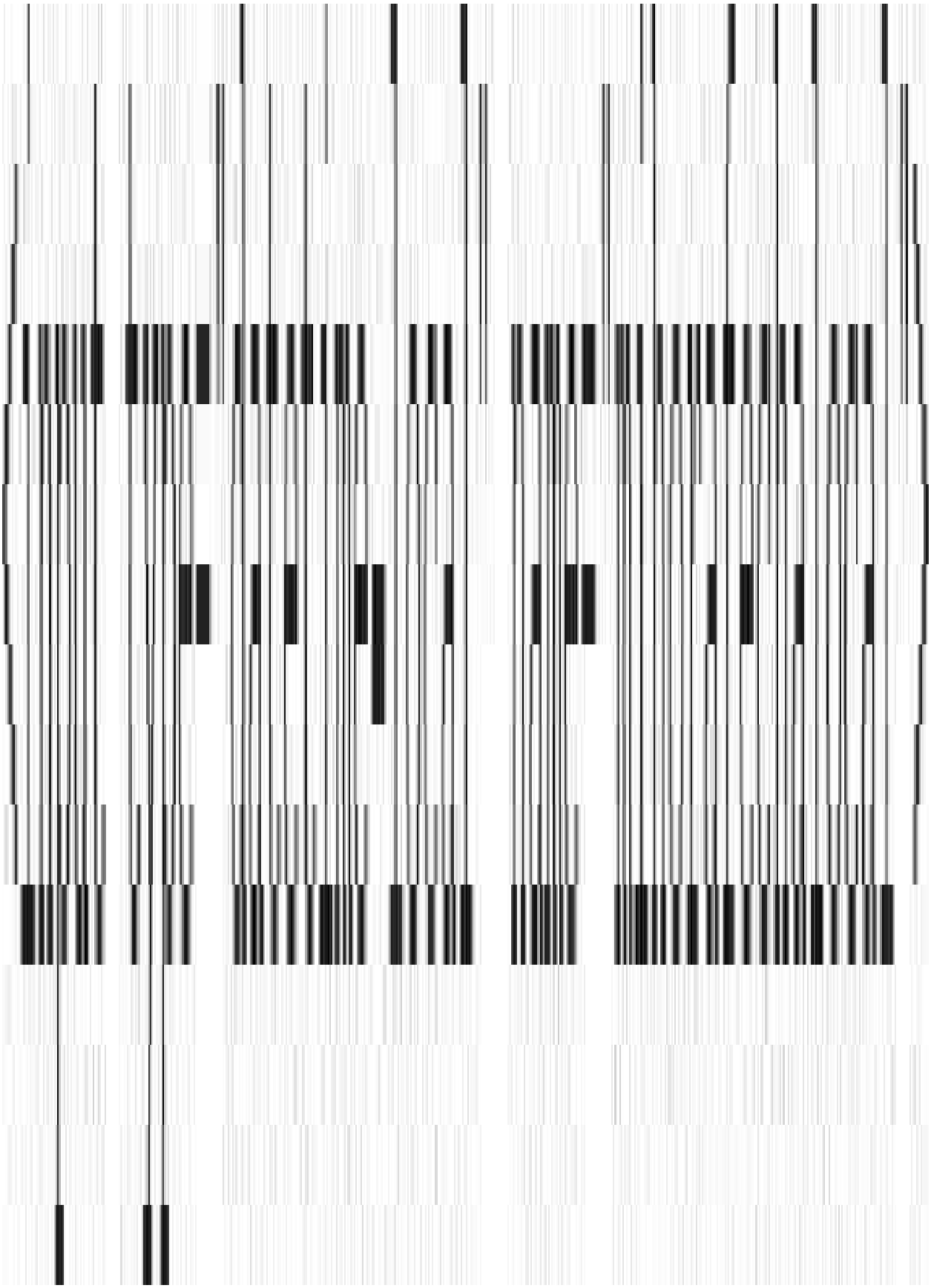


Código 5-15

Declaración del tipo datetime.

5.2.12 Tipo datetime-local

El tipo datetime-local es igual al tipo datatime sin zona horaria.

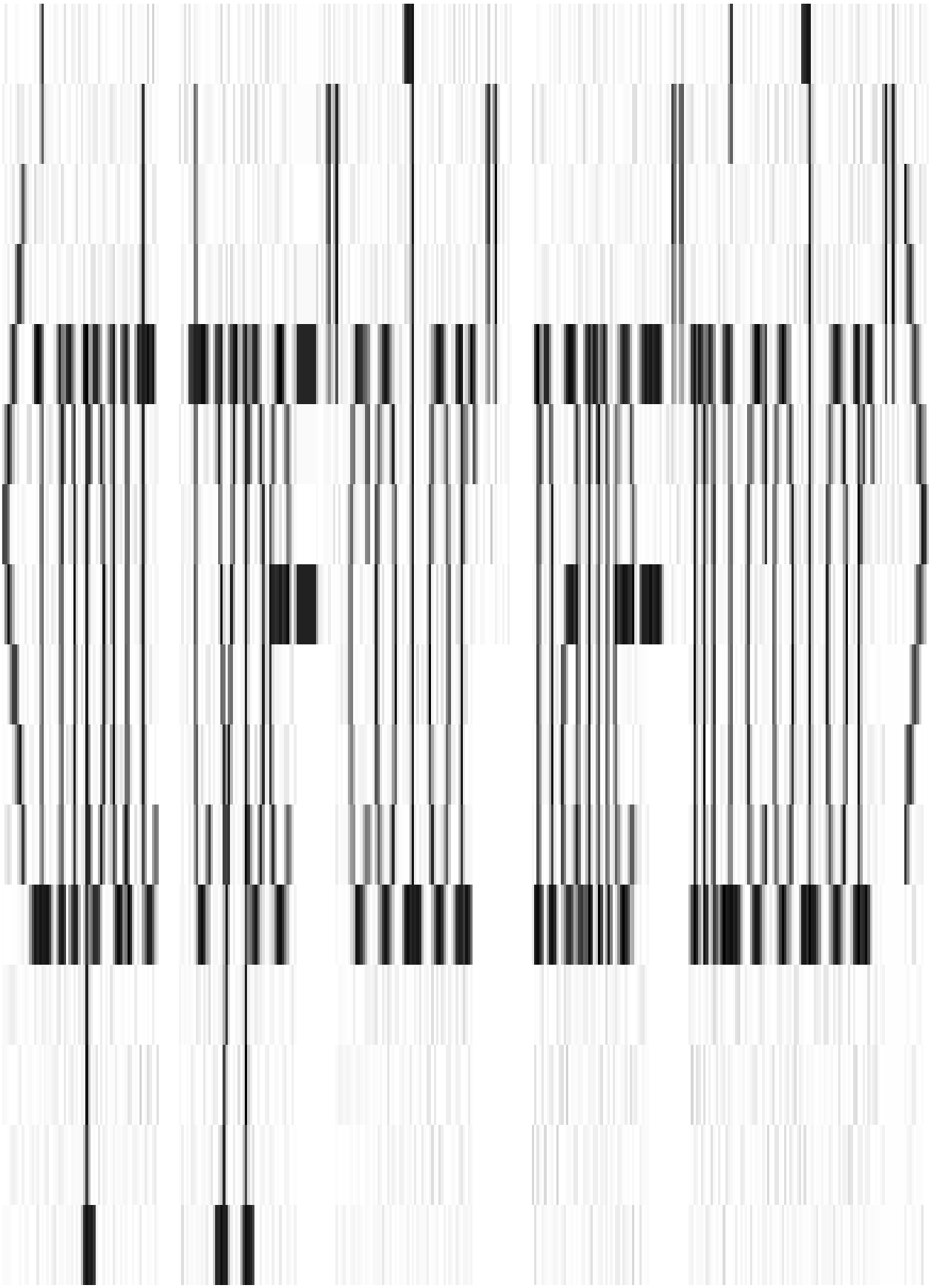


Código 5-16

Declaración del tipo datetime-local.

5.2.13 Tipo color

Además de los tipos de fecha y hora, existe también un tipo que proporciona una interfaz predefinida que permite a los usuarios seleccionar un color. Por lo general, el valor esperado para este campo es un número hexadecimal, tal como #00FF00.



Código 5-17

Declaración del tipo color.

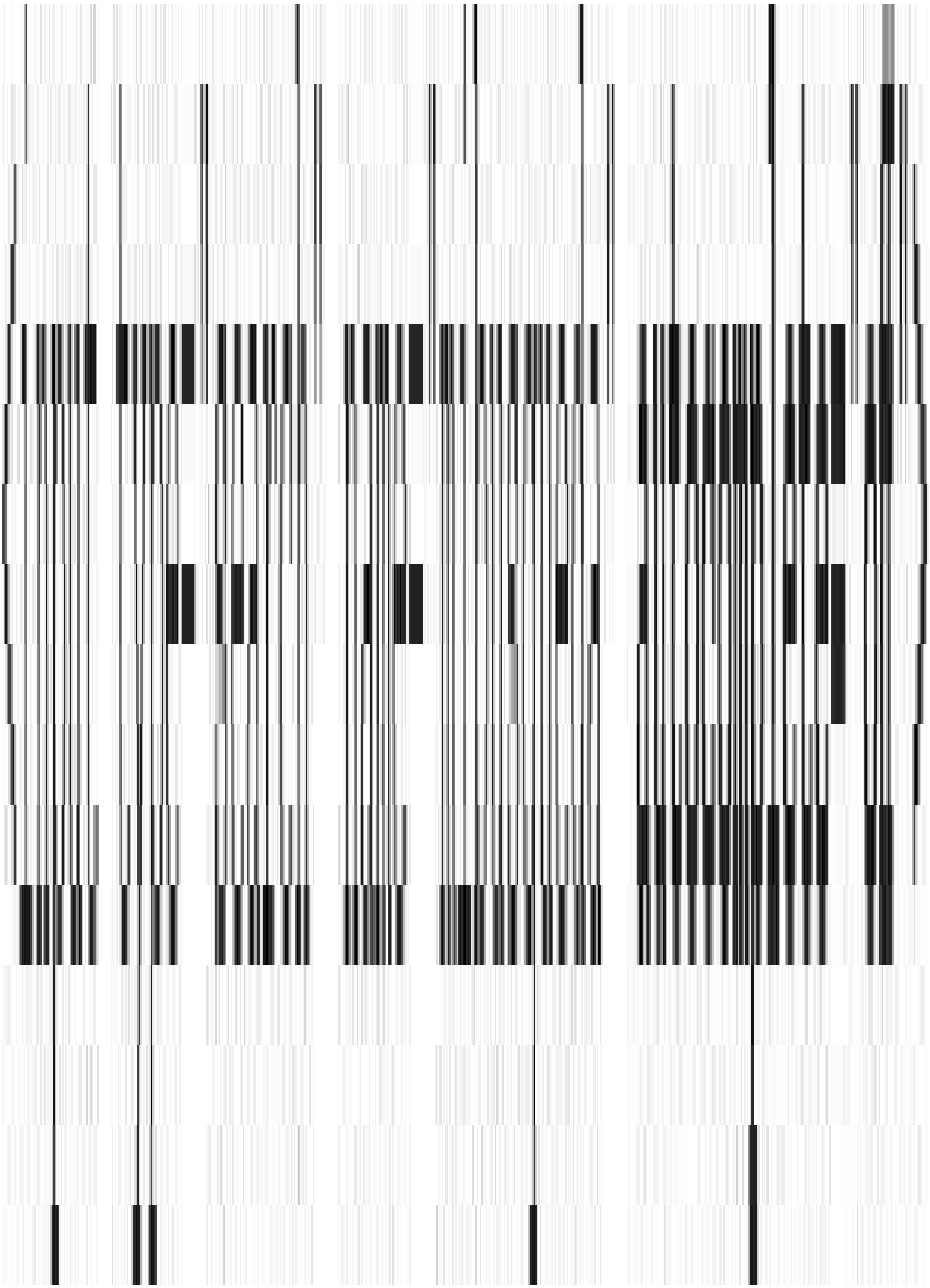
No hay una interfaz estándar especificada por HTML5 para el tipo color, pero es posible que en el futuro sea adoptada por los navegadores una tabla regular con un conjunto de colores básicos.

5.3 Nuevos atributos

Algunos tipos de entrada requieren la ayuda de atributos como min, max y step, de los que ya hemos hablado, para llevar a cabo sus tareas. Otros tipos de entrada requieren la asistencia de atributos para mejorar su rendimiento o para determinar su importancia en el proceso de validación. Los siguientes son los nuevos atributos incorporados en HTML5 para estos propósitos:

5.3.1 Atributo autocomplete

El atributo autocomplete es un antiguo atributo que se ha estandarizado. Puede tomar dos valores: on y off, y el valor por defecto es on. Cuando el valor es off, el elemento <input> tiene la función de autocompletar desactivada, y por tanto no muestra los textos de las entradas anteriores como posibles valores. También puede ser usado en el elemento <form> para afectar a todos los elementos del formulario.



Código 5-18

Uso del atributo autocomplete.

5.3.2 Atributos novalidate y formnovalidate

Una de las características de los formularios en HTML5 es su capacidad de validación integrada. Por defecto, todos los formularios se validan a menos que el atributo novalidate esté presente. Este atributo booleano es específico para la etiqueta <form>. Cuando se incluye, el formulario se envía sin validación.

A veces, el proceso de validación se requiere solo en algunas circunstancias. Por ejemplo, cuando la información insertada debe ser salvada para reanudar el trabajo más tarde. En casos como éste, podemos confiar en otro atributo booleano específico para los elementos <input> llamado formnovalidate.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Formularios</title>
</head>
<body>
  <section>
    <form name="miformulario" method="get" action="file.php">
      <input type="email" name="micorreoelectronico">
      <input type="submit" value="Enviar">
      <input type="submit" value="Guardar" formnovalidate>
    </form>
  </section>
</body>
</html>
```

Código 5-19

Envío de un formulario sin validación con el atributo formnovalidate.

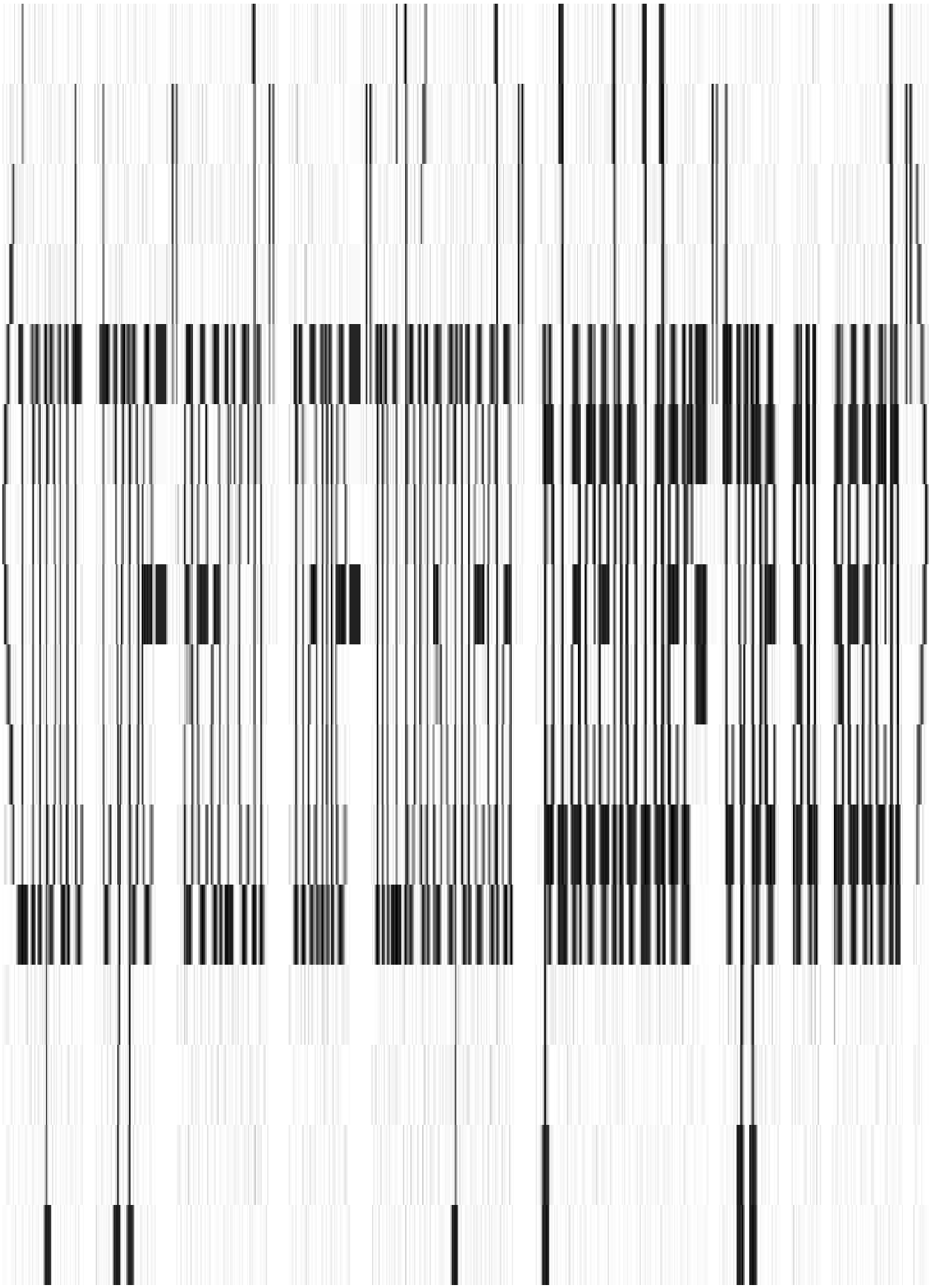
Importante

No es el propósito de este libro enseñar las expresiones comunes. Para obtener más información acerca de este tema, por favor visite nuestro sitio web y siga los enlaces de este capítulo.

En el ejemplo del Código 5-19, el formulario va a ser validado, pero se ofrece un segundo botón de envío con el atributo formnovalidate para que el formulario pueda ser enviado sin pasar por el proceso de validación. El botón Enviar requiere un valor válido para el campo micorreoelectronico, pero el botón Guardar no.

5.3.3 Atributo placeholder

El atributo placeholder representa una breve pista o referencia, que puede ser una palabra o una frase, destinada a ayudar al usuario a introducir la entrada correcta. El valor de este atributo es mostrado por los navegadores en el interior del campo como un texto de vista previa que desaparece cuando el elemento es seleccionado.

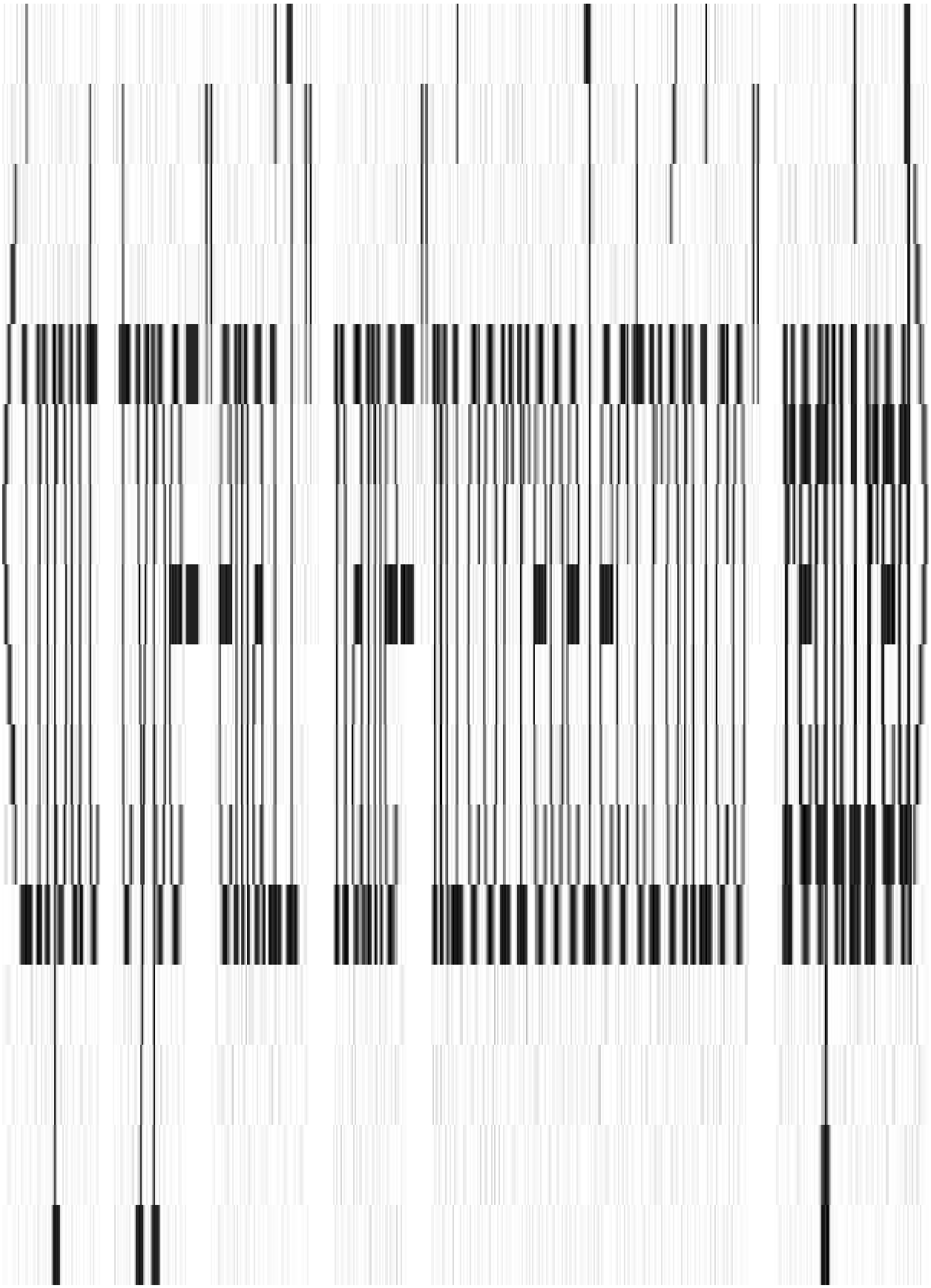


Código 5-20

Uso del atributo placeholder.

5.3.4 Atributo required

El atributo booleano required establece que el formulario no se envíe si el campo está vacío. Por ejemplo, al usar un campo del tipo email para recibir una dirección de correo electrónico sin usar el atributo required, el navegador invalida el campo si se introduce un valor que no corresponde a una dirección de correo electrónica válida, pero en cambio sí permite que el campo esté vacío. Cuando se incluye el atributo required en el código de la entrada, la entrada será válida solo cuando el y usuario rellena el campo y el valor intorducido se ajusta a los requisitos de su tipo.

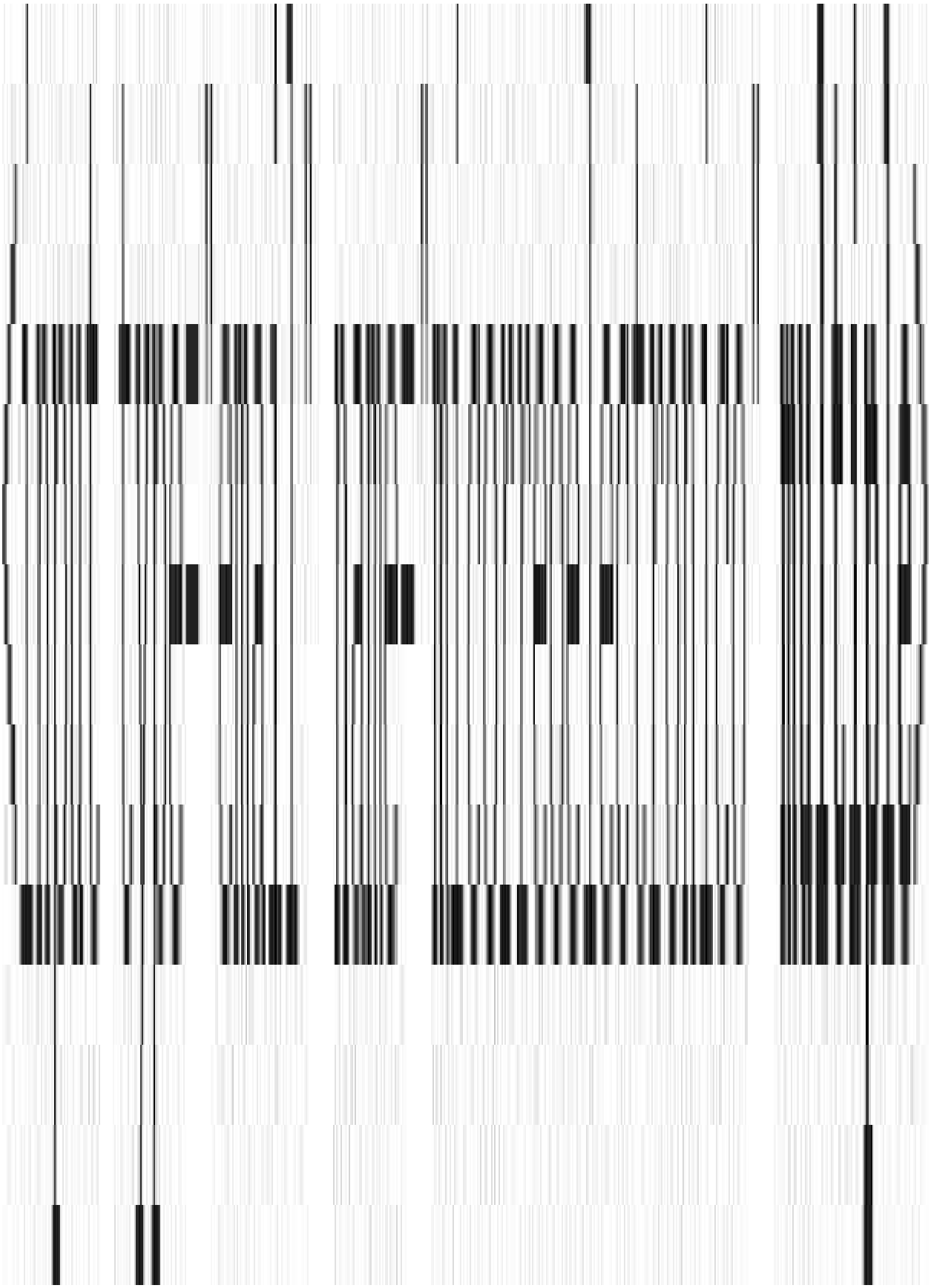


Código 5-21

Declaración de la entrada email como un campo obligatorio.

5.3.5 Atributo multiple

El atributo multiple es otro booleano que se puede utilizar en algunos tipos de entrada (por ejemplo, email o file) para permitir múltiples entradas en un mismo campo. Loa valores insertados tienen que ser separados por comas para ser válidos.



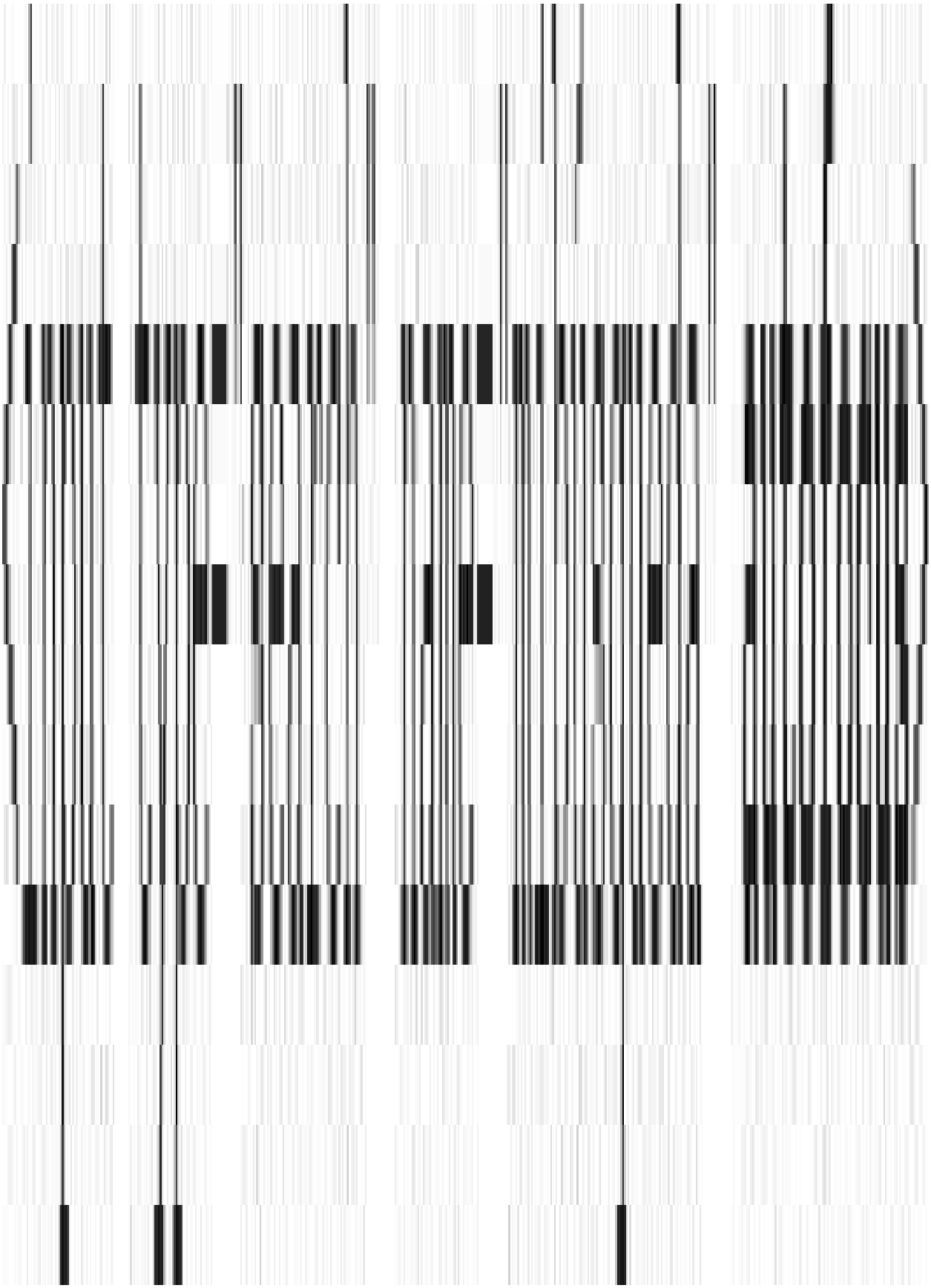
Código 5-22

La declaración del campo email como campo múltiple.

El Código 5-22 permite insertar múltiples valores separados por comas. Cada valor debe ser validado por el navegador como una dirección de correo electrónico.

5.3.6 Atributo autofocus

La función autofocus ha sido aplicada por la mayoría de los desarrolladores que utilizan el método de Javascript focus(). Este método era eficaz pero forzaba el foco sobre el elemento seleccionado, incluso cuando el usuario ya estaba utilizando otro. Este comportamiento era irritante pero imposible de evitar, hasta ahora. El atributo autofocus centra la página web sobre el elemento seleccionado pero teniendo en cuenta la situación actual; no moverá el enfoque cuando ya ha sido establecido en otro elemento.

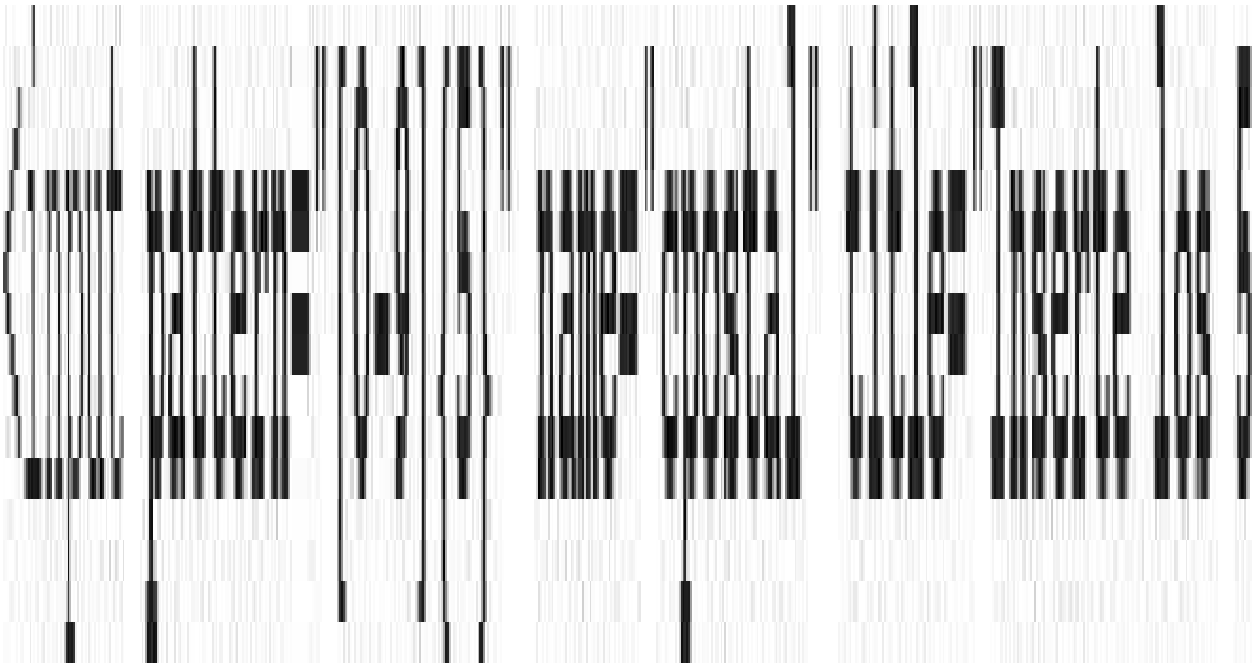


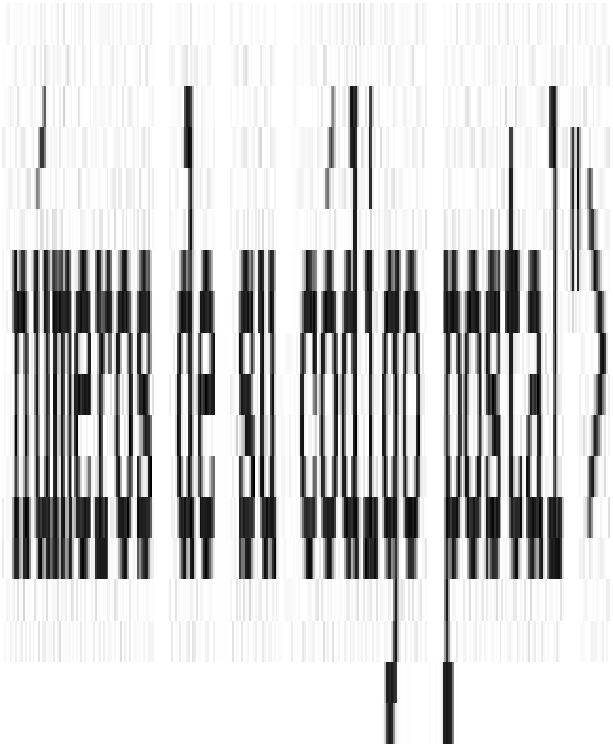
Código 5-23

Uso del atributo autofocus.

5.3.7 Atributo pattern

El atributo pattern aumenta las alternativas de validación previstas por defecto. Utiliza expresiones comunes para personalizar las reglas de validación. Algunos de los tipos de entrada mencionados validan determinados tipos de cadenas, pero supongamos, por ejemplo, que desea validar un código postal que consta de 5 números. No hay ningún tipo de entrada predeterminado para este tipo de entrada. El atributo pattern le permite en ese caso crear su propio tipo de entrada para comprobar los valores. También puede incluir un atributo title para personalizar el mensaje de error.





Código 5-24

Personalizar los tipos con el atributo pattern.

5.3.8 Atributo form

El atributo form permite declarar elementos de una formulario fuera de la etiquetas <form>. Hasta ahora, para construir un formulario tenía que escribir las etiquetas de apertura y cierre <form> y luego declarar todos los elementos del formulario entre ellos. En HTML5, utilizando el atributo form, podemos insertar los elementos en cualquier lugar y después hacer referencia al formulario al que pertenecen por su nombre.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Formularios</title>
</head>
<body>
  <nav>
    <input type="search" name="mibúsqueda" form="miformulario">
  </nav>
  <section>
    <form name="miformulario" method="get" action="file.php">
      <input type="text" name="minombre">
      <input type="submit" value="Enviar">
    </form>
  </section>
</body>
</html>
```

Código 5-25

La declaración de los elementos de formulario en cualquier lugar del código.

5.4 Nuevos elementos de los formularios

En páginas anteriores hemos hablado de los nuevos tipos y atributos disponibles para los elementos <input> introducidos en HTML5, y ahora conoceremos los más relevantes de los nuevos elementos HTML que han sido incorporados para mejorar o ampliar las posibilidades de creación de formularios en línea.

5.4.1 El elemento <datalist>

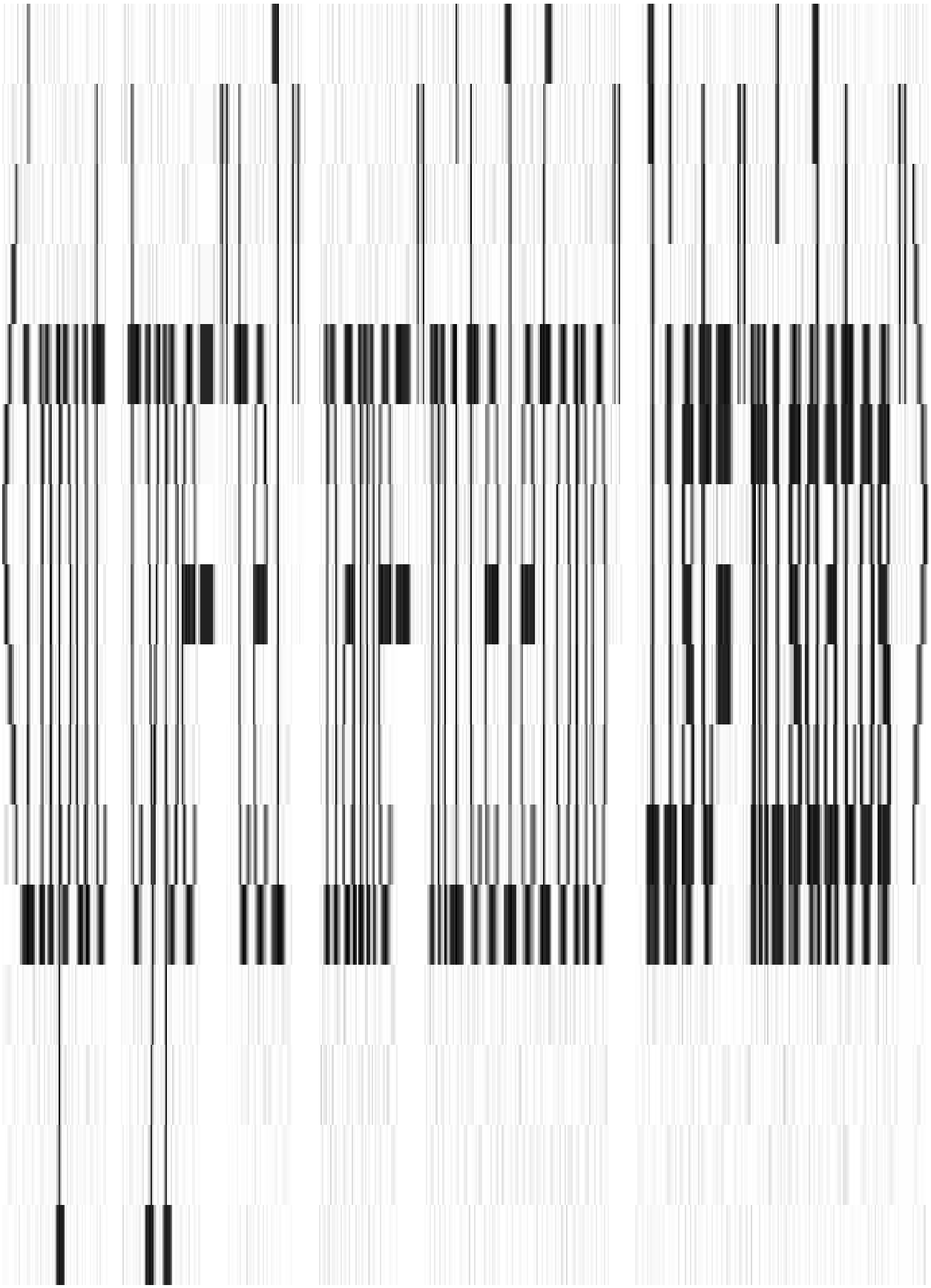
El elemento <datalist> construye una lista de textos que, con la ayuda del atributo list, se utilizará más tarde como sugerencias para un campo de entrada.

```
<datalist id="misdatos">

    <option value="12312312" label="Telefono 1">

    <option value="45645645" label="Telefono 2">

</datalist>
```

Código 5-26

Construcción de una lista de opciones.

Una vez está declarado el elemento <datalist>, el paso siguiente es hacer referencia a la lista de elementos desde un elemento <input> mediante el atributo list. El valor requerido para este atributo es el valor del atributo id para el elemento <datalist>. El elemento del ejemplo siguiente muestra los posibles valores que el usuario puede elegir.

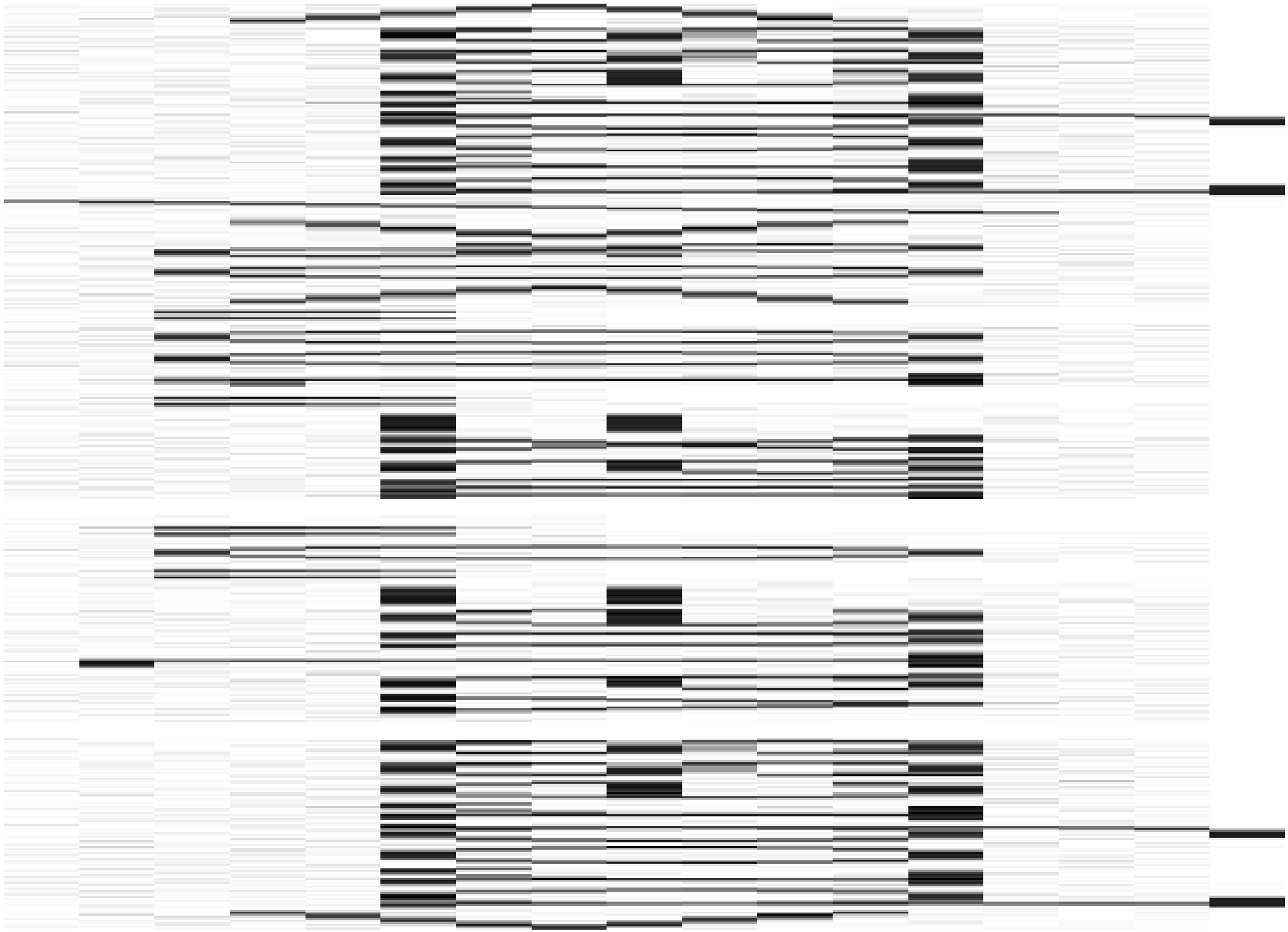


Código 5-27

Sugerir valores con el atributo lista.

5.4.2 El elemento <progress>

Este elemento no es específicamente de formularios, pero como representa el progreso de una tarea y por lo general las tareas se inician y se procesan a través de formularios, puede ser considerado un elemento de formulario. El elemento <progress> utiliza dos atributos para establecer su estado y sus límites. El atributo value indica cuánto de la tarea se ha completado, y max declara el que debe ser alcanzado para que la tarea se considere completada.

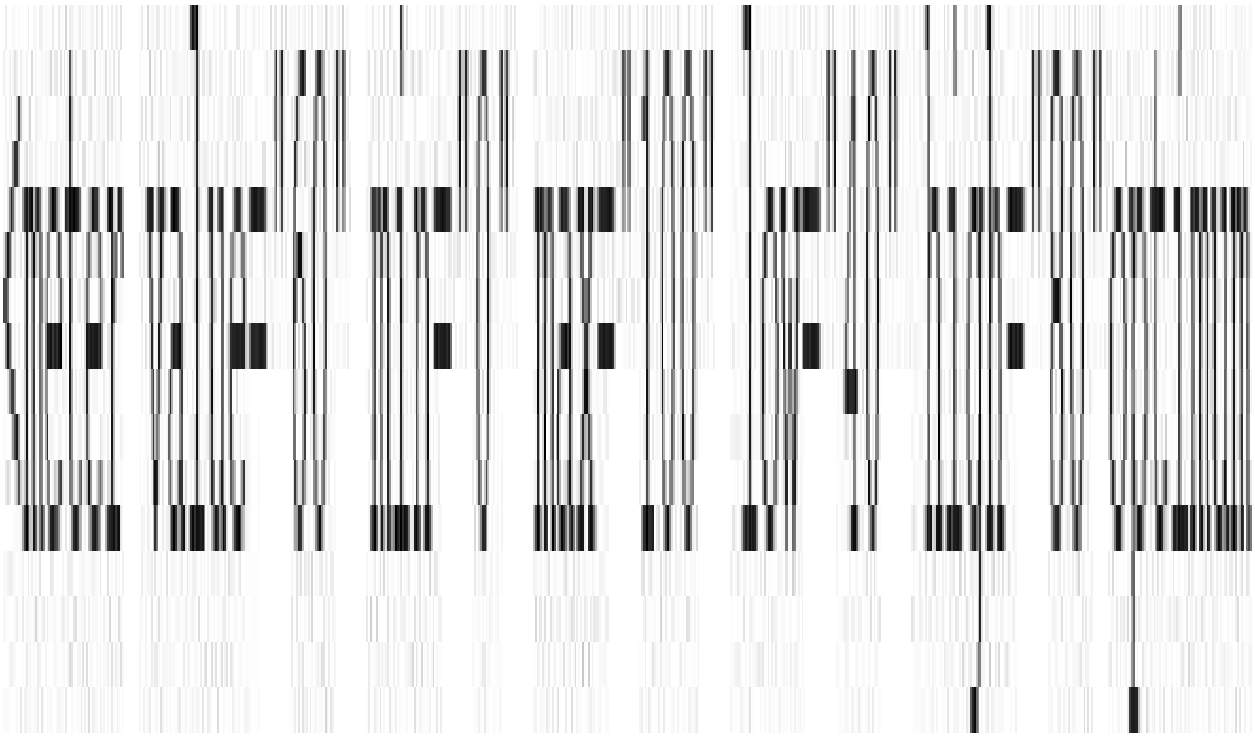


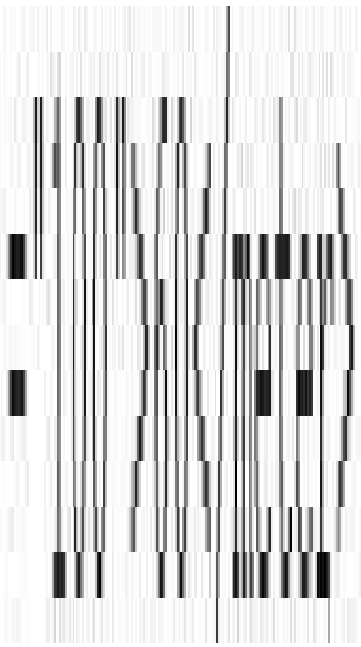
Código 5-28

Uso del elemento <progress>.

5.4.3 El elemento <meter>

Similar a <progress>, el elemento <meter> se utiliza para mostrar una escala, pero no de progresión. Pretende ser una representación de un rango conocido, por ejemplo el uso de ancho de banda. El elemento <meter> tiene varios atributos asociados. Los atributos min y max establecien los límites del rango, valor determina el valor medido, y low, high y optimum se utilizan para segmentar el rango en secciones diferenciadas y establecer la posición óptima.





Código 5-29

Uso del elemento <meter>.

El Código 5-29 genera una barra que muestra un nivel de 60 en una escala de 0 a 100 (de acuerdo con los valores declarados por los atributos value, min y max). El color de la barra generada por este elemento en pantalla dependerá de los niveles establecidos por los atributos low, high y optimum. En este caso será amarillo.

5.4.4 El elemento <output>

El elemento output representa el resultado de un cálculo. Por lo general ayuda a mostrar el resultado de los valores procesados por los elementos del formulario. El atributo for permite asociar al elemento <output> con el elemento que participa en el cálculo, pero el elemento es referido y modificado desde el código Javascript. La sintaxis de este elemento es <output>value</output>.

5.5 Nueva pseudo-clases

Ya, hemos mencionado que es posible personalizar la apariencia de los elementos inválidos del formulario. CSS3 ofrece nuevas pseudo-clases que nos permiten declarar nuestros propios estilos no solo para los elementos inválidos, sino también para los elementos válidos, obligatorios, opcionales e incluso de rango limitado, como pueden ser elementos <input> con el tipo range. Ahora, cada vez que se establece una nueva condición para un elemento, CSS es responsable de mostrar el nuevo estado en la pantalla.

5.5.1 valid e invalid

Estas pseudo-clases representan cualquier elemento <input> con valor, sea éste válido o no lo sea.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Formularios</title>
  <style>
    :valid{
      background: #EEEEFF;
    }
    :invalid{
      background: #FFEEEE;
    }
  </style>
</head>
<body>
  <section>
    <form name="miformulario" method="get" action="file.php">
      <input type="email" name="micorreoelectronico">
      <input type="submit" value="Enviar">
    </form>
  </section>
</body>
</html>
```

Código 5-30

Uso de las pseudo-clases :valid e :invalid.

El formulario del Código 5-30 incluye un elemento <input> para correos electrónicos. Cuando el contenido del elemento es válido para un tipo email, la pseudo-clase :valid asigna un fondo azul al campo. Tan pronto como el contenido se convierte en no válido, la pseudo-clase :invalid cambia el color de fondo a rojo.

5.5.2 optional y required

Estas pseudo-clases representan cualquier elemento del formulario, haya sido declarado como obligatorio o no.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Formularios</title>
  <style>
    :required{
      border: 2px solid #990000;
    }
    :optional{
      border: 2px solid #009999;
    }
  </style>
</head>
<body>
  <section>
    <form name="miformulario" method="get" action="file.php">
      <input type="text" name="minombre">
      <input type="text" name="miapellido" required>
      <input type="submit" value="Enviar">

    </form>
  </section>
</body>
</html>
```

Código 5-31

Uso de las pseudo-clases :required y :optional.

El ejemplo del Código 5-31 incluye dos campos de entrada: minombre y miapellido. El primero es opcional, pero miapellido es obligatorio. Las pseudo-clases colorean el borde de estos campos de acuerdo a su condición.

5.5.3 in-range y out-of-range

Estas pseudo-clases representan cualquier elemento cuyos valores tengan limitaciones de rango (por ejemplo, un elemento <input> con el tipo número).

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Formularios</title>
  <style>
    :in-range{
      background: #EEEEFF;
    }
    :out-of-range{
      background: #FFEEEE;
    }
  </style>
</head>
<body>
  <section>
    <form name="miformulario" method="get" action="file.php">
      <input type="number" name="mynumber" min="0" max="10">
      <input type="submit" value="Enviar">
    </form>
  </section>
</body>
</html>
```

Código 5-32

Uso de las pseudo-clases :in-range y :out-of-range.

Importante

Las pseudo-clases pueden aplicarse solo a elementos específicos usando selectores CSS y atributos id o class (por ejemplo, #Micorreoelectronico:invalid{}).

En el ejemplo anterior insertamos un campo de entrada del tipo number para probar las pseudo-clases mencionadas. Cuando el valor introducido en el elemento es menor que 0 o mayor que 10, el fondo se tiñe de color rojo, pero en cuanto se introduce en un valor dentro del rango especificado, el fondo es azul.

5.6 Formularios API

Como habrá notado, hay diferentes maneras de sacar provecho de la validación de formularios en HTML5. Es posible utilizar los tipos de entrada que requieren una validación por defecto, como mail, convertir un campo del tipo text en un campo requerido con el atributo required, o incluso el uso de tipos especiales, como pattern para personalizar los requisitos de validación. Sin embargo, cuando se trata de mecanismos de validación más complejos, tales como la combinación de campos o la comprobación de los resultados de un cálculo anterior, la única opción es personalizar el proceso de validación mediante los nuevos formularios API.

5.6.1 SetCustomValidity()

Los navegadores que soportan HTML5 muestran un mensaje de error cuando el usuario intenta enviar un formulario con un campo no válido. Puede crear mensajes para sus propios requisitos de validación utilizando el método setCustomValidity(mensaje).

Con este método se establece un error personalizado que mostrará un mensaje cuando se envíe el formulario. Cuando se proporciona un mensaje nulo, el error se borra.

```
 <!DOCTYPE html>
<html lang="es">
<head>
  <title>Formularios</title>
  <script>
    var nombre1, nombre2;
    function iniciar(){
      nombre1 = document.getElementById("nombre");
      nombre2 = document.getElementById("apellido");
      nombre1.addEventListener("input", validación);
      nombre2.addEventListener("input", validación);
      validación();
    }
    function validación(){
      if(nombre1.value == '' && nombre2.value == ''){
        nombre1.setCustomValidity('Inserte al menos un nombre');
        nombre1.style.background = '#FFDDDD';
        nombre2.style.background = '#FFDDDD';
      }else{
        nombre1.setCustomValidity('');
        nombre1.style.background = '#FFFFFF';
        nombre2.style.background = '#FFFFFF';
      }
    }
    addEventListener("load", iniciar);
  </script>
</head>
<body>
  <section>
    <form name="registration" method="get" action="file.php">
      <label for="nombre">Nombre: </label>
      <input type="text" name="nombre" id="nombre">
      <label for="apellido">Apellido: </label>
      <input type="text" name="apellido" id="apellido">
      <input type="submit" value="Iniciar sesión">
    </form>
  </section>
</body>
</html>
```

Código 5-33

Configurar errores personalizados.

Conceptos básicos

Como ya sabe, cada elemento HTML es un objeto (de hecho, se reproduce como un objeto en una estructura interna creada por el navegador). La mayoría de los atributos de elementos son accesibles desde Javascript usando el nombre del atributo como una propiedad del elemento (por ejemplo, nombre1.value). En el ejemplo anterior, los valores de los campos de entrada se recuperan utilizando la propiedad value, pues los valores insertados en los campos de entrada se almacenan como valor del atributo value.

El documento de Código 5-33 presenta una validación compleja. Se crean dos campos de entrada para nombre y apellido del usuario. Sin embargo, el formulario solo es inválido cuando ambos campos están vacíos, es decir, el usuario puede introducir solo un nombre o un apellido para validar la entrada. En un caso como éste es imposible utilizar el atributo required porque no sabemos qué campo de entrada elegirá el usuario. solo es posible crear un mecanismo de validación eficaz para este escenario usando código Javascript y errores personalizados. Este código se pone en marcha cuando el evento load se dispara. Se llama a la función iniciar() para controlar el evento. Esta función crea referencias para los dos elementos <input> y añade detectores a ambos para el evento input.

La función validacion(), establecida como detector para los mencionados eventos, se ejecuta cada vez que el usuario tipea en los campos para agregar o eliminar caracteres.

Como los elementos <input> están vacíos cuando se carga el documento, hay que establecer una condición inválida para impedir que el usuario envíe el formulario antes de haber escrito al menos un nombre. Por esta razón, la función validacion() también es llamada al final de la función iniciar() para comprobar esta condición. Si ambos nombres son cadenas de texto vacías, se envía el error y el color de fondo de ambos elementos cambia a rojo. Sin embargo, si esa condición deja de cumplirse porque se inserta al menos uno de los nombres, el error se borra y los fondos recuperan el color blanco.

Recuerde que el único cambio que se produce durante el procesamiento de este código es la modificación del color de fondo. El mensaje declarado para el error con SetCustomValidity() solo será visible cuando se intente enviar el formulario.

Conceptos básicos

Es posible declarar diversas variables separadas por una coma en la misma línea. En el Código 5-33, declaramos dos variables globales: nombre1 y nombre2, para poder acceder a los valores de cualquier función.

Podría haberse evitado la declaración, ya que, como hemos mencionado antes, las variables declaradas dentro de funciones sin el operador var tienen alcance global.

Hágalo usted mismo

Para facilitar la prueba, el código Javascript ha sido incluido en el código del ejemplo anterior. Para probar el ejemplo solo tiene que copiar el Código 5-33 en un archivo HTML vacío y abrir el archivo en su navegador.

5.6.2 El evento invalid y el método CheckValidity()

Cuando un usuario envía un formulario, se activa un evento si es detectado un elemento no válido. El evento se llama invalid y se refiere al elemento que produce el error. Es posible registrar un controlador de eventos para personalizar la respuesta:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Formularios</title>
  <script>
    var form;
    function iniciar(){
      var boton = document.getElementById("Enviar");
      boton.addEventListener("click", enviarlo);
      form = document.querySelector("form[name='información']");
      form.addEventListener("invalid", validación, true);
    }
    function validación(e){
      var elem = e.target;
      elem.style.background = '#FFDDDD';
    }
    function enviarlo(){
      var valid = form.checkValidity();
      if(valid){
        form.submit();
      }
    }
    addEventListener("load", iniciar);
  </script>
</head>
<body>
  <section>
    <form name="información" method="get" action="file.php">
      <label for="nickname">Apodo: </label>
      <input pattern="[A-Za-z]{3,}" name="nickname" id="apodo"
 maxlength="10" required>
      <label for="micorreoelectronico">Email: </label>
      <input type="email" name="micorreoelectronico"
 id="micorreoelectronico" required>
      <input type="button" id="Enviar" value="Iniciar sesión">
    </form>
  </section>
</body>
</html>
```

Código 5-34

La creación de nuestro propio sistema de validación.

En el Código 5-34 se crea un nuevo formulario con dos campos de entrada para pedir un apodo y una dirección de correo electrónico. La entrada nickname tiene tres atributos de validación: el atributo pattern, que solo admite un mínimo de 3 caracteres de la A a la Z (mayúsculas y minúsculas), el atributo maxlength, que limita la entrada a un máximo de 10 caracteres, y el atributo required, que invalida el campo si está vacío. La entrada email, por otra parte, tiene una limitación natural debido a su tipo y un atributo required. Lo que hace el código Javascript con este formulario es simple: cuando el usuario hace clic en el botón Iniciar sesión, se lanza un evento invalid por cada campo no válido y el color de fondo de esos campos cambia a rojo gracias a la función validación().

Importante

Los procesos de registrar controladores de eventos o añadir detectores de eventos, tales como addEventListener(), siempre envían un objeto que hace referencia al evento. En objeto es enviado como un atributo a la función que hace de controlador. Tradicionalmente, se utiliza la variable e para almacenar este valor, pero puede utilizar cualquier nombre que desee. El objeto proporciona propiedades con información importante sobre el evento. La propiedad target, por ejemplo, devuelve una referencia al elemento que disparó el evento. Vamos a ver otras propiedades de eventos y más ejemplos a lo largo del libro.

Pero vayamos más allá: el código comienza a correr cuando el evento load se activa después de que el documento se acaba de cargar. La función iniciar() se ejecuta y se añaden detectores al botón Iniciar sesión y al elemento <form>.

El detector para el evento click del botón Iniciar sesión se añade como lo hicimos antes, pero el detector para el evento invalid es diferente porque debe controlar la validación para todo el formulario y tiene que ser añadido al elemento <form> y no a cada elemento dentro de éste. Para lograrlo hay que capturar todos los eventos de nivel inferior en el árbol HTML, y por eso el último parámetro del método addEventListener() utilizado para el evento invalid se configura como true. Como resultado, aunque el detector ha sido añadido al elemento <form>, responde a los eventos disparados por los elementos que están dentro del formulario.

Cuando el evento invalid se activa, se llama a la función validacion() para cambiar el color de fondo del campo no válido. Este evento será disparado por una entrada no válida cuando se envíe el formulario y luego será capturado por el detector del elemento <form>. Para determinar cuál es el elemento inválido, se almacena una referencia al evento en la variable e y se lee el valor de la propiedad target, que a su vez devuelve una referencia al elemento que generó el evento. Con esta referencia, la última instrucción de la función validacion() cambia el color de fondo del elemento correspondiente.

Volviendo a la función iniciar(), aún tenemos un detector por analizar. Para tener un control absoluto sobre la presentación del formulario y el momento de la validación, se ha creado un botón simple en vez de un botón del tipo submit. La función enviarlo(), añadida antes como controlador del evento click para este elemento, se ejecuta cuando el botón es pulsado. El uso del método CheckValidity() obliga al navegador a validar y solo envía el formulario usando el método submit() cuando no hay más condiciones inválidas. El código Javascript del último ejemplo controla al máximo el proceso de validación, la personalización de cada aspecto y el comportamiento del navegador.

En el último ejemplo se aplicó el método querySelector() para obtener una referencia al formulario usando un selector CSS. Este selector busca un elemento <form> con el valor información para el atributo name. Consulte el Capítulo 2 para más información sobre los selectores CSS y el Capítulo 4 para revisar el método querySelector().

5.6.3 Validación en tiempo real con ValidityState

Al abrir el archivo del documento del Código 5-34 en el navegador, notará que no existe una validación en tiempo real. Los campos se validan solo cuando se envía el formulario. Para que el proceso de validación personalizado sea más práctico, pueden usarse las diversas propiedades proporcionadas por el objeto ValidityState.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Formularios</title>
  <script>
    var form;
    function iniciar(){
      var boton = document.getElementById("Enviar");
      boton.addEventListener("click", enviarlo);
      form = document.querySelector("form[name='información']");
      form.addEventListener("input", comprobar);
       }
    function validación(e){
      var elem = e.target;
      elem.style.background = '#FFDDDD';
    }
    function enviarlo(){
      var valid = form.checkValidity();
      if(valid){
        form.submit();
      }
    }
    function comprobar (e){
      var elem = e.target;
      if(elem.validity.valid){
        elem.style.background = '#FFFFFF';
      }else{
        elem.style.background = '#FFDDDD';
      }
    }
    addEventListener("load", iniciar);
    </script>
</head>
<body>
  <section>
    <form name="información" method="get" action="file.php">


      <label for="nickname">Apodo: </label>
      <input pattern="[A-Za-z]{3,}" name="nickname" id="apodo"
maxlength="10" required>
      <label for="micorreoelectronico">Email: </label>
      <input type="email" name="micorreoelectronico"
id="micorreoelectronico" required>
      <input type="button" id="Enviar" value="Iniciar sesión">
   </form>
 </section>
```

Código 5-35

Validación en tiempo real.

En el Código 5-35, se añade al formulario un nuevo detector del evento input. Cada vez que el usuario introduce o modifica el contenido de un campo, la función comprobar() se ejecuta para controlar el evento.

Además, la función comprobar() saca provecho de la propiedad target para crear una referencia al elemento que generó el evento y controla su validez mediante la comprobación del estado valid con la propiedad validity en la construcción elem.validity.valid. Esta propiedad hace referencia al objeto ValidityState, como veremos a continuación.

El valor del estado valid será true si el elemento es válido y false si no lo es. Con esta información, podemos cambiar el color del fondo para el elemento que lanzó el evento de entrada. Se aplicará color blanco a los campos válidos y el rojo para los no válidos.

Con este simple cambio, ahora cada vez que el usuario modifica el valor de cualquier elemento de la forma, éste será comprobado y su condición se reflejará en la pantalla en tiempo real.

5.6.4 Restricciones de validez

En el ejemplo del Código 5-35, se comprobó el estado valid, que es una característica del objeto ValidityState que devuelve el estado de validez de un elemento teniendo en cuenta todos los otros estados posible de validez. Todas las condiciones son válidas, entonces la propiedad valid devuelve el valor true.

Hay ocho estados de validez para diferentes condiciones:

valueMissing: El valor de este estado es true cuando el atributo required se declara y el campo de entrada está vacío.

typeMismatch: Es true cuando la sintaxis de la entrada no se ajusta al tipo especificado, por ejemplo, si el texto introducido en un campo del tipo email no es una dirección de correo electrónico.

patternMismatch: Es true cuando la entrada no coincide con el patrón proporcionado.

tooLong: Es true cuando se ha declarado el atributo maxlength y la entrada es mayor que el valor especificado para aquel.

rangeUnderflow: Es true cuando el atributo min es declarado y la entrada es menor que el valor especificado para el atributo.

rangeOverflow: Es true cuando el atributo max es declarado y la entrada es mayor que el valor especificado para el atributo.

stepMismatch: Es true cuando el atributo step es declarado y su valor no corresponde con el valor de atributos tales como min, max y value.

CustomError: Es true cuando se establece un error personalizado, por ejemplo, con el método SetCustomValidity().

Para acceder al objeto ValidityState y comprobar los estados de validez, hay que utilizar la propiedad validity con la sintaxis element.validity.status, donde status es cualquiera de los valores enumerados anteriomente. Puede usar estas propiedades para saber exactamente qué desencadenó el error en el formulario, como en el ejemplo siguiente:

```
function enviarlo(){
  var elem = document.getElementById("apodo");
  var valido = form.checkValidity();
  if(valido){
    form.submit();
  }else if(elem.validity.patternMismatch || elem.validity.
 valueMissing){
    alert('El apodo debe tener al menos tres caracteres');
  }
}
```

Código 5-36

Usar ValidityStatus para mostrar un mensaje de error personalizado.

En el Código 5-36, la función SendIt() se ha modificado para incluir un control personalizado. El formulario se valida por el método CheckValidity(), y si es válido, se envía con submit(). De lo contrario, se comprueban los estados de validez patternMismatch y valueMissing para la entrada nickname y se muestra un mensaje de error cuando de ellos uno o ambos devuelven el valor true.

Conceptos básicos

La declaración if puede ser concatenada usando la construcción else if (con un espacio entre las dos palabras). En el ejemplo anterior, si la primera condición es falsa, se evalúa la declaración if después de else. Puede seguir añadiendo if indefinidamente para buscar más condiciones si es necesario.

Hágalo usted mismo

Remplace la función Enviarlo() en el documento del Código 5-35 con la nueva función presentada en el Código 5-36 y abra el archivo HTML en su navegador.

6 Vídeo y audio

6.1 Vídeo con HTML5

Una de las características más discutidas de HTML5 era el procesamiento de vídeo, no por las nuevas herramientas proporcionadas por HTML5 para este fin, sino más bien porque el vídeo se había convertido en la pieza central de Internet y todo el mundo esperaba el soporte nativo de los navegadores. Parecía que todos, salvo aquellos que estaban desarrollando nuevas tecnologías para la web, eran conscientes de la importancia de los vídeos para la red.

Pero ahora que finalmente hay un soporte nativo e incluso un estándar que permite construir aplicaciones de procesamiento de vídeo para diversos navegadores, nos damos cuenta de que era más complicado de lo que habíamos imaginado. Desde códecs hasta problemas de consumo de recursos, las razones de que antes no se implementara vídeo eran mucho más complejas que los códigos necesarios para hacerlo.

A pesar de las complicaciones, HTML5 finalmente introdujo un elemento para insertar y reproducir archivos de vídeo en un documento HTML. El elemento <video> utiliza etiquetas de apertura y cierre y solo necesita unos pocos parámetros para llevar a cabo su función. La sintaxis es extremadamente simple y el único atributo, src, es obligatorio.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Reproductor de vídeo</title>
</head>
<body>
  <section>
    <video src="http://minkbooks.com/content/trailer.mp4" controls>
    </video>
  </section>
</body>
</html>
```

Código 6-1

Uso del elemento <video>.

En teoría, el Código 6.1 debería ser más que suficiente. Pero como ya hemos dicho, las cosas son un poco más complicadas de lo esperado. En primer lugar, hay que proporcionar al menos dos archivos con formatos de vídeo diferentes. Esto es porque aunque el elemento <video> y sus atributos son estándar, no existe un formato de vídeo estándar. El problema es que algunos navegadores son compatibles con un grupo de códecs que otros no (y al revés), y el códec utilizado en el formato MP4 (el único soportado por los navegadores importantes, como Safari e Internet Explorer) tiene licencia comercial.

Las opciones más comunes hoy en día son OGG y MP4, pero WebM está tratando de abrirse paso y convertirse en el estándar. Estos formatos son contenedores de vídeo y audio. OGG contiene vídeo Theora y audio de códec Vorbis, MP4 contiene H.264 para vídeo y AAC para audio, y utiliza codec de vídeo WebM VP8 y códec de audio Vorbis. En la actualidad, OGG y WebM son compatibles con Mozilla Firefox, Google Chrome y Opera, mientras que MP4 funciona en Safari, Internet Explorer y Google Chrome.

6.1.1 El elemento <video>

Evitemos esas complicaciones y disfrutemos de la simplicidad del elemento <video> por un momento. Este elemento tiene varios atributos para establecer sus propiedades y configuración por defecto. Los atributos width y height, como los de un tradicional elemento <img>, declaran las dimensiones del elemento o ventana de reproducción en píxeles. El tamaño del vídeo se ajustará automáticamente para encajar en estas dimensiones, pero no están destinados a comprimir o estirar el vídeo. Deben usarse para limitar el área ocupada por el vídeo y preservar la coherencia del diseño. El atributo src especifica la fuente. Este atributo puede ser sustituido por el elemento <source> y su propio atributo src para declarar varias fuentes para diferentes formatos de vídeo.

Hágalo usted mismo

Cree un nuevo archivo HTML vacío con un nombre y una extensión .html (por ejemplo, video.html), copie el documento HTML del Código 6-2 en el archivo y ábralo en diferentes navegadores para ver el elemento <video> en funcionamiento.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Reproductor de vídeo</title>
</head>
<body>
  <section>
    <video width="720" height="400" controls>
      <source src="http://minkbooks.com/content/trailer.mp4">
      <source src="http://minkbooks.com/content/trailer.ogg">
    </video>
  </section>
</body>
</html>
```

Código 6-2

Creación de un reproductor de vídeo con controles por defecto.

Importante

Los navegadores requieren que los vídeos sean enviados por el servidor con correspondiente el tipo MIME. Cada archivo tiene un tipo MIME asociado para indicar el formato de su contenido. Por ejemplo, el tipo MIME de un archivo HTML es text/html. Los servidores ya están configurados para los formatos de vídeo más comunes, pero por lo general no para los nuevos formatos como OGG o WebM. Cómo incluir el nuevo tipo MIME dependerá de qué tipo de servidor que tiene. Una manera fácil de hacerlo es añadir una nueva línea al archivo .htaccess. La mayoría de los servidores ofrecen esta configuración en la carpeta raíz de cada sitio web. La sintaxis correspondiente es Addtype MIME/tipo extensión (por ejemplo, AddType video/ogg).

En el Código 6-2, el elemento <video> es expandido. Ahora, dentro de las etiquetas del elemento, hay dos elementos <source>. Estos elementos proporcionan diferentes fuentes de vídeo para el navegador para elegir. El navegador leerá las etiquetas <source> y decidirá qué archivo se debe reproducir de acuerdo a los formatos soportados (en este caso, MP4 u OGG).

6.1.2 Atributos del elemento <video>

Hay un atributo en la etiqueta <video> utilizada en los Códigos 6-1 y 6-2 que probablemente le habrá llamado la atención. El atributo controls, que es uno de los atributos específicos disponibles para este elemento, muestra los controles de vídeo proporcionados por los navegadores. Cada navegador activará su propia interfaz, que permitirá al usuario iniciar el video, hacer una pausa o saltar a un fotograma específico, entre otras cosas.

Además del atributo controls, el elemento <video> también dispone de otros atributos:

autoplay: Cuando este atributo está presente, el navegador reproduce automáticamente el vídeo tan pronto como puede.

loop: Si se especifica este atributo, el navegador reinicia el vídeo al llegar al final. del mismo.

poster: Este atributo proporciona una dirección URL de una imagen mientras se espera que comience la reproducción del vídeo.

preload: Este atributo puede tomar tres valores: none, metadata o auto. El primer valor indica que el vídeo no debe ser almacenado en el caché, por lo general con el fin de minimizar el tráfico innecesario. El segundo valor, metadata, recomienda al navegador buscar información sobre el vídeo, como las dimensiones, la duración, el primer fotograma, etc. El tercer valor,

auto, es el establecido por defecto e indica que se descargue el archivo tan pronto como sea posible.

```
<!DOCTYPE html>
<html lang="es">

<head>
  <title>Reproductor de vídeo</title>
</head>
<body>
  <section>
    <video width="720" height="400" preload controls loop
           poster="http://minkbooks.com/content/poster.jpg">
      <source src="http://minkbooks.com/content/trailer.mp4">
      <source src="http://minkbooks.com/content/trailer.ogg">
    </video>
  </section>
</body>
</html>
```

Código 6-3

Usar los atributos de <video>.

El documento HTML del Código 6-3 precarga el vídeo adecuado para el navegador, ofrece controles por defecto, muestra una imagen en lugar del vídeo hasta que el vídeo comienza a reproducirse, y reproduce el vídeo una y otra vez.

6.1.3 Formatos de vídeo

Como hemos explicado anteriormente, en la actualidad, no existe un formato estándar de audio y vídeo para la Web. Hay varios contenedores y diferentes códecs disponibles, pero ninguno ha sido adoptado de forma demasiado extendida, y no parece que los proveedores de navegadores hayan alcanzado un consenso que les permita hacerlo en un futuro próximo.

Los contenedores más comunes son OGG, MP4, FLV y WebM, el más nuevo, propuesto por Google:

OGG: Códec de vídeo Theora y códec de audio Vorbis.

MP4: Códec de vídeo H.264 y códec de audio AAC.

FLV: Códec de vídeo VP6 y códec de audio MP3. También es compatible con H.264 y AAC.

WEBM: Códec de vídeo VP8 y codec de audio Vorbis.

Los códecs utilizados para OGG y WebM son libres, pero los códecs utilizados para MP4 y FLV están restringidos por patentes, lo que significa que si desea utilizar MP4 y FLV para sus aplicaciones, tendrá que pagar por ello, aunque algunas restricciones han sido revocadas para las aplicaciones gratuitas.

Una cuestión importante es que, en la actualidad, Safari e Internet Explorer no son compatibles con la tecnología libre y ambos trabajan solo con MP4. Sin embargo, Internet Explorer ha anunciado la inclusión del códec de vídeo VP8 en el futuro.

Presentamos una lista de los códecs compatibles con cada navegador:

Mozilla Firefox: Compatible con códecs de vídeo Theora y VP8 y códec de audio Vorbis (OGG y WebM).

Google Chrome: Compatible con códecs de vídeo Theora y VP8 y audio Vorbis. También es compatible con el códec de vídeo H.264 y el códec de audio AAC (OGG, WebM y MP4).

Ópera: Compatible con códecs de vídeo Theora y VP8 y audio Vorbis (OGG y WebM).

Safari: Compatible con el códec de vídeo H.264 y el códec de audio AAC (MP4).

Internet Explorer: Compatible con el códec de vídeo H.264 y el códec de audio AAC (MP4).

En el futuro, el apoyo a formatos abiertos como WEBM hará las cosas más fáciles, pero probablemente no habrá un formato estándar en los próximos dos o

tres años y, por tanto, tendremos que considerar diferentes alternativas de acuerdo con la naturaleza de nuestra aplicación y negocios.

6.2 Audio con HTML5

El audio no es tan popular como el vídeo en la Web. Es posible grabar un video con una cámara personal que genere millones de visitas en sitios web como www.youtube.com pero lograr el mismo resultado con un archivo de audio es casi imposible. Sin embargo, el audio aún está presente y su mercado está creciendo a través de programas de radio y difusión en la Web. HTML5 ofrece un nuevo elemento para reproducir audio en un documento HTML. El elemento, por supuesto, es <audio> y comparte todas las características con el elemento <video>.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Reproductor de video</title>
</head>
<body>
  <section id="reproductor">
    <audio src="http://minkbooks.com/content/beach.mp3" controls>
    </audio>
  </section>
</body>
</html>
```

Código 6-4

Uso del elemento <audio>.

6.2.1 El elemento <audio>

El elemento <audio> funciona de la misma manera y comparte varios atributos con el elemento <video>:

src: Este atributo especifica la URL del archivo que se va a reproducir. Al igual que sucede con elemento <video>, src normalmente es reemplazado por el elemento <source> para proporcionar diferentes formatos de audio para el navegador para elegir.

controls: Activa la interfaz que proporciona por defecto cada navegador.

autoplay: Cuando este atributo está presente, el navegador reproduce el audio automáticamente tan pronto como puede.

loop: Si se especifica este atributo, el navegador reproduce el audio una y otra vez.

preload: Este atributo puede tomar tres valores: none, metadata o auto. El primer valor indica que el audio no debe ser almacenado en el caché, por lo general con el objetivo de reducir al mínimo el tráfico innecesario. El segundo valor, metadata, recomienda al navegador buscar información sobre el material (por ejemplo, su duración). El tercer valor, auto, es el establecido por defecto y le indica al navegador que descargue el archivo tan pronto como sea posible.

De nuevo tenemos que hablar de có decs y, de nuevo, el código HTML 6-4 debería ser más que suficiente, pero no lo es. MP3 se encuentra bajo licencia comercial, por lo que no es soportado por navegadores como Mozilla Firefox u Opera. Vorbis (el códec de audio del contenedor OGG) es soportado por estos navegadores, pero no por Safari ni por Internet Explorer. Por tanto, una vez más es necesario utilizar el elemento <source> para proporcionar al navegador al menos dos formatos para elegir.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Reproductor de vídeo</title>
</head>
<body>
  <section id="reproductor">
    <audio id="medio" controls>
      <source src="http://minkbooks.com/content/beach.mp3">
      <source src="http://minkbooks.com/content/beach.ogg">
    </audio>
  </section>
</body>
</html>
```

Código 6-5

Creación de un reproductor de audio con controles por defecto compatible con diversos navegadores.

El documento del Código 6-5 reproduce la música en cualquier navegador con sus controles por defecto. Los que no pueden reproducir MP3, reproducirán OGG y viceversa. Recuerde que el uso de MP3, así como pasaba con el vídeo MP4, está restringido por licencias comerciales, por lo que solo podrá utilizarlo en circunstancias específicas determinadas por la licencia.

El soporte para los códecs de audio libres (como Vorbis) está en proceso de expansión, pero se necesitará tiempo para convertir un formato desconocido en un estándar.

6.3 Subtítulos

El subtitulado es un método simple para proporcionar información adicional y ampliar el alcance de los medios de comunicación. Esta función muestra el texto en la pantalla, mientras que el material se está reproduciendo. Se ha utilizado en la televisión y las diferentes formas de distribución de vídeo durante décadas, pero siempre ha sido difícil de repetir en la Web.

HTML5 ha solucionado este problema con la incorporación de un nuevo elemento y una sencilla API para este propósito.

6.3.1 El elemento <track>

El elemento <track> permite a los desarrolladores incorporar subtítulos a los medios de comunicación web. Funciona con los elementos <video> y <audio> y acepta varios formatos de archivo para mostrar la información (aunque la especificación declara el formato WebVTT como el oficial). Gracias a este nuevo elemento estamos en condiciones de añadir subtítulos o mostrar información adicional para nuestros vídeos y pistas de audio.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Subtítulos de vídeo</title>
</head>
<body>
  <section>
    <video width="720" height="400" controls>
      <source src="trailer.mp4">
      <source src="trailer.ogg">
      <track src="subtitulos.vtt" srclang="en" default>
    </video>
  </section>
</body>
</html>
```

Código 6-6

Adición de subtítulos con el elemento <track>.

Importante

El elemento <track> no permite la construcción de aplicaciones de origen transversal, por lo que tendrá que subir todos los archivos a su propio servidor (usando cualquier programa de FTP como FileZilla) y ejecutar todo bajo el mismo dominio (incluido el documento HTML, los videos y los subtítulos). Sin embargo, puede evitar esto usando la tecnología de CORS y el atributo crossorigin, tal como se explica en el Capítulo 10.

El elemento <track> tiene que ser incluido como un elemento hijo del elemento <video> o <audio>. Proporciona los siguientes atributos para especificar la fuente, el tipo y cómo se mostrarán los subtítulos en la pantalla:

src: Este atributo declara la URL del archivo que contiene el texto de los subtítulos. El formato de este archivo puede ser cualquiera de los soportados por los navegadores (la especificación declara el formato WebVTT como el oficial para este elemento).

srclang: Este atributo declara el idioma del texto. Trabaja con los mismos valores que el atributo lang del elemento <html> estudiado en el Capítulo 1.

default: Los archivos multimedia pueden incluir varios elementos <track> para diferentes propósitos y lenguajes. El atributo default declara cuál se muestra por defecto. Si solo se proporciona un elemento <track>, se puede usar este atributo para activar los subtítulos.

label: Este atributo proporciona un título para la pista o elemento <track>. En el caso de múltiples elementos <track>, este atributo ayudará a los usuarios a encontrar el correcto.

kind: Este atributo declara el tipo de contenido de la pista. Los valores posibles son subtitles (para subtítulos), captions (por lo general para los títulos que representan sonidos), descriptions (para la síntesis de audio), chapters (para la navegación entre capítulos) y metadate (para información adicional, no mostrada en la pantalla). El valor por defecto es subtitles.

En el ejemplo del Código 6-6 se declara un solo elemento <track>. El idioma de la fuente establecido es el Inglés, y se incluyó el atributo default para establecer esta pista por defecto y activar los subtítulos. También se declaró como fuente un archivo en formato WebVTT (subtitles.vtt). WebVTT es Tracks Text Web Video y lo que hace este formato es proporcionar un formato estándar para los subtítulos. Un archivo .vtt es solo un archivo de texto con una estructura específica:

WEBVTT

00:02.000 --> 00:07.000

¡Presentamos el elemento <track>!

00:10.000 --> 00:15.000

Esto es solo un ejemplo.

00:17.000 --> 00:22.000

Podrá usar varios canales de manera simultánea

00:22.000 --> 00:25.000

para proporcionar textos en varios idiomas.

00:27.000 --> 00:30.000

¡Hasta pronto!

Código 6-7

Archivo WebVTT.

El Código 6-7 muestra la estructura básica de un archivo WebVTT. La primera línea con el texto WEBVTT es obligatoria, igual que la línea en blanco entre las declaraciones. Estas declaraciones se denominan cues (entradas), y requieren la sintaxis minutos:segundos.milisegundos para indicar los tiempos de inicio y final. Éste es un formato rígido; hay que respetar la estructura que se muestra en el ejemplo del Código 6-7 y declarar cada parámetro con el mismo número de dígitos (dos para los minutos, dos para los segundos y tres para los milisegundos).

Importante

WebVTT utiliza codificación UTF-8. Al guardar el archivo de texto mediante un editor como el Bloc de notas de Windows, seleccione UTF-8 en las opciones de codificación.

Hágalo usted mismo

Cree un archivo HTML con el Código 6-6. Utilizando su editor de texto, cree un archivo WebVTT con las declaraciones del Código 6-7 y el nombre

subtitulos.vtt. Suba estos archivos y los archivos de vídeo a su servidor usando cualquier programa de FTP como FileZilla y abra el documento HTML en su navegador. Puede descargar nuestros vídeos con las direcciones URL de los ejemplos anteriores, utilizar su propio vídeo, u obtenerlos desde www.minkbooks.com/content/.

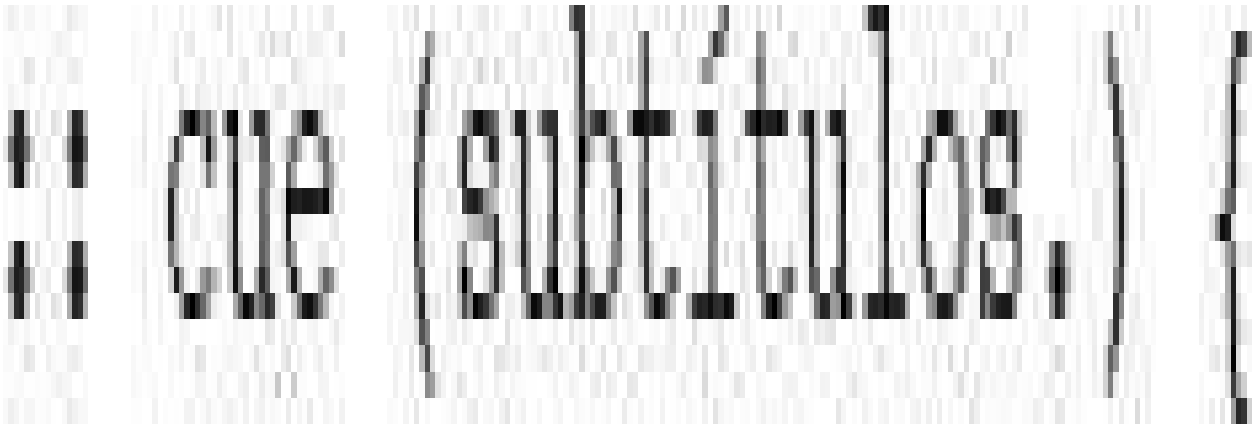
Las cues (las líneas de texto de un archivo de subtítulos) pueden incluir etiquetas especiales, tales como <b>, <i>, <u>, <v> y <c>. Las primeras tres, como en HTML, son para el énfasis, mientras que la etiqueta <v> declara qué voz representa el texto y la etiqueta <c> permite ofrecer estilos personalizados mediante CSS.

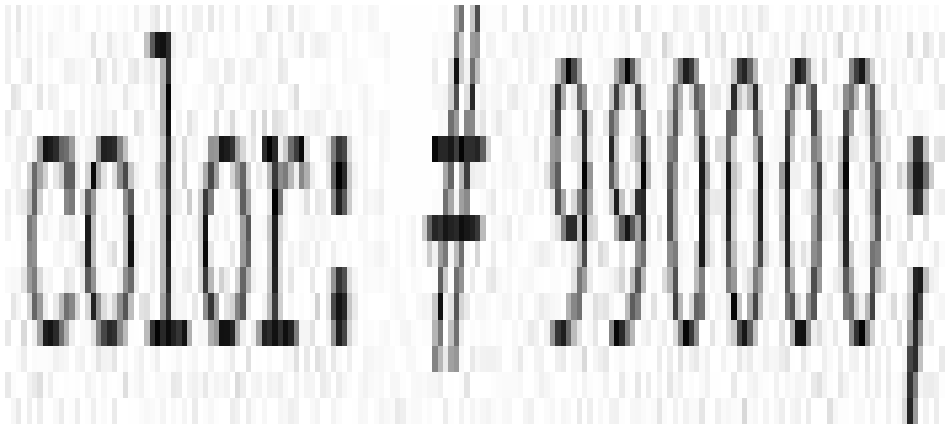
```
WEBVTT

00:02.000 --> 00:07.000

<i>Bienvenido</i>

¡Presentamos el elemento &lt;track&gt;!

00:10.000 --> 00:15.000

<v Roberto>Esto es solo un <c.subtitulo>ejemplo</c>.

00:17.000 --> 00:22.000

<v Martín>Podrá usar varios canales de manera simultánea

00:22.000 --> 00:25.000

<v Martín>para proporcionar textos en varios idiomas.

00:27.000 --> 00:30.000

<b>¡Hasta pronto!<b>
```

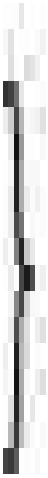
Código 6-8

Etiquetas incluidas en un fichero WebVTT.

WebVTT utiliza un pseudo elemento para hacer referencia a las entradas de texto. Este elemento, llamado ::Cue, puede tomar un selector entre paréntesis para una clase específica. En el ejemplo del Código 6-8, la clase se llama captions (<c. captions >), así que el selector CSS debe ser identificado como ::cue(c.subtitulo), Como en el ejemplo siguiente:


::cue(.subtitulos) {


color: #990000;


}

Código 6-9

CSS para WebVTT.

Importante

Hay solo un grupo reducido de propiedades de CSS, como color, fondo y fuente, disponibles para esta pseudo-clase, y el resto son ignoradas. Al momento de escribir este libro, la etiqueta <c> aún no ha sido implementada en los navegadores. Para comprobar el estado actual de esta especificación y encontrar actualizaciones para los ejemplos anteriores, vaya al sitio web www.minkbooks.com/updates/.

WebVTT también proporciona la capacidad de alinear y posicionar cada entrada de texto utilizando los parámetros y valores siguientes:

align: Alinea la entrada de texto respecto al centro del espacio cubierto por los objetos multimedia. Los valores posibles son start, middle y end.

vertical: Establece una orientación vertical y ordena las entradas de texto en función de dos valores: rl (de derecha a izquierda) o lr (de izquierda a derecha).

position: Ajusta la posición de la entrada de texto en las columnas. El valor se puede expresar como un porcentaje o un número de 0 a 9. La posición se declarada env función de la orientación.

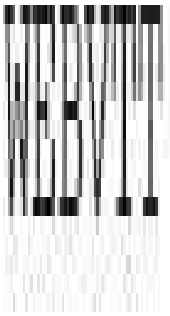
line: Este parámetro ajusta la posición de la entrada en filas. El valor se puede expresar como un porcentaje o un número de 0 a 9. En una orientación horizontal, la posición de la declaración es vertical, y viceversa. Los números positivos establecen la posición desde un lado y los negativos desde el otro, dependiendo de la orientación.

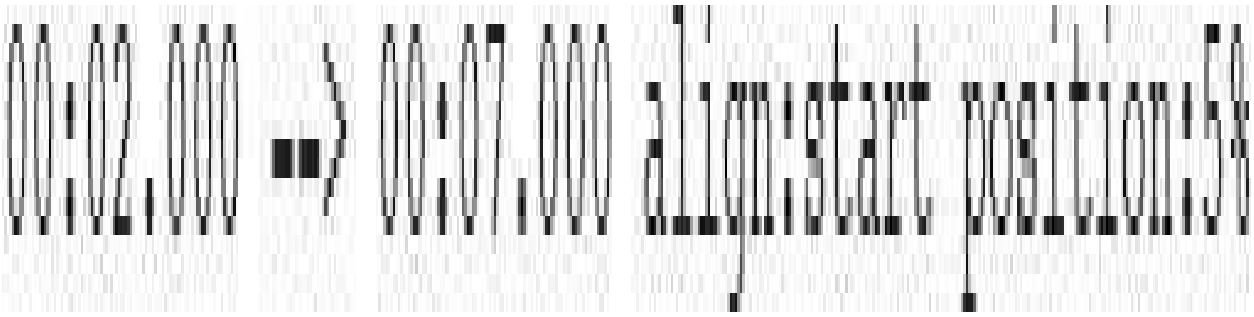
size: Este parámetro ajusta el tamaño de la entrada de texto. El valor puede ser declarado como porcentaje y representa el porcentaje del ancho del objeto multimedia.

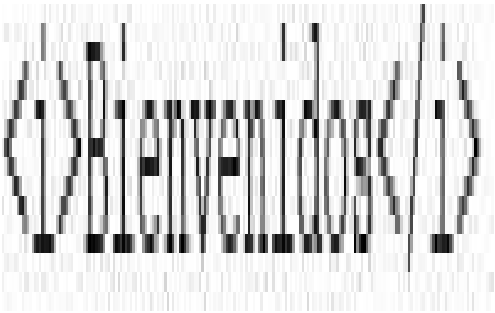
Hágalo usted mismo

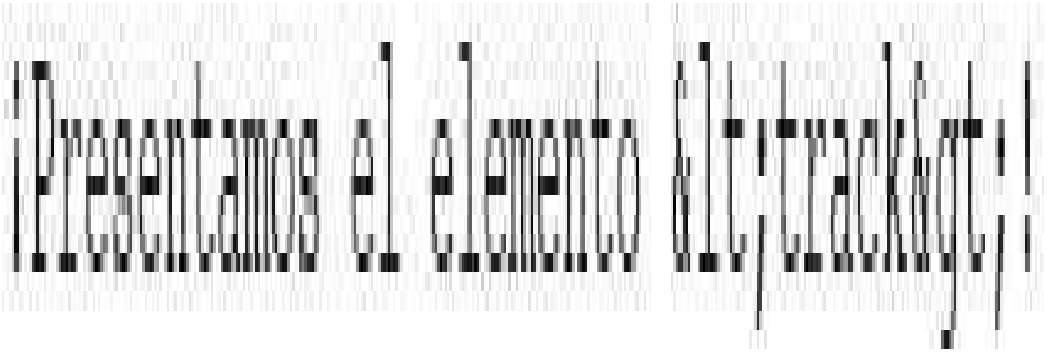
Usando el archivo WebVTT creado en el Código 6-7, intente combinar diferentes parámetros en las mismas entradas de texto para ver los efectos disponibles para este formato.

Estos parámetros y sus correspondientes valores se declaran al final de la primera línea de la entrada, que indica el tiempo, separados por dos puntos. Puede insertar múltiples declaraciones para una misma entrada, tal como se muestra en el ejemplo siguiente:


WEBVTT


00:02.000 --> 00:07.000 align:start position:5%


Bienvenidos


¡Presentamos el elemento <track>!

Código 6-10

Configuración de las señales.

6.4 Programar un reproductor multimedia

Si ha probado los ejemplos anteriores en distintos navegadores, se habrá dado cuenta de que el diseño gráfico del panel de control del reproductor es diferente en cada caso.

Cada navegador tiene sus propios botones y barras de progreso, e incluso sus propias características. Esta situación podría ser aceptable en algunas circunstancias, pero en un entorno profesional, donde cada detalle cuenta, es absolutamente necesario tener el control sobre todo el proceso y proporcionar un diseño consistente en todos los dispositivos y aplicaciones.

HTML5 ofrece nuevos eventos, propiedades y métodos para manipular e integrar vídeo y audio. Ahora es posible crear un reproductor de vídeo o de audio propio y proporcionar las características deseadas usando HTML, CSS y Javascript. Los elementos multimedia ahora son parte del documento.

Importante

Los eventos, propiedades y métodos de esta API son los mismos para vídeo y audio. Para usar los códigos que estudiaremos a continuación para archivos de audio, solo hay que sustituir el elemento <video> por el elemento <audio> en el documento HTML y proporcionar los correspondientes archivos de audio para la fuente.

6.4.1 Diseño de un reproductor de vídeo

Todos los reproductores de vídeo necesitan un panel de control con algunas características básicas. En el documento HTML del Código 6-11, se añade después del elemento <video> un elemento <nav> que contiene dos botones, un deslizador y una barra de progreso.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Reproductor de vídeo</title>
  <link rel="stylesheet" href="reproductor.css">
  <script src="reproductor.js"></script>
</head>
<body>
  <section id="reproductor">
    <video id="media" width="720" height="400">
      <source src="trailer.mp4">
      <source src="trailer.ogg">
    </video>
    <nav>
      <div id="botones">
        <input type="button" id="play" value="Reproducir">
        <input type="button" id="mute" value="Silencio">
      </div>
      <div id="barra">
        <div id="progreso"></div>
      </div>
      <div id="control">
        <input type="range" id="volume" min="0" max="1" step="0.1"
 value="0.6">
      </div>
      <div class="clear"></div>
    </nav>
  </section>
</body>
</html>
```

Código 6-11

Documento HTML de nuestro reproductor de vídeo.

Este documento también incluye recursos de los dos archivos externos. Uno de esos archivos es reproductor.css para los siguientes estilos CSS:

```
body{
  text-align: center;
}
header, section, footer, aside, nav, article, figure, figcaption,
 hgroup{
  display: block;
}
#reproductor{
  width: 720px;
  margin: 20px auto;
  padding: 10px 5px 5px 5px;
  background: #999999;
  border: 1px solid #666666;
  border-radius: 10px;
}
#play, #mute{
  padding: 2px 10px;
  width: 100px;
  border: 1px solid #000000;
  background: #DDDDDD;
  font-weight: bold;
  border-radius: 10px;
}
nav{
  margin: 5px 0px;
}
#botones{
  float: left;
  width: 135px;
  height: 50px;
  padding-left: 5px;
}
#barra{
  float: left;
  width: 400px;
  height: 16px;
  padding: 2px;
  margin: 2px 5px;
  border: 1px solid #CCCCCC;
  background: #EEEEEE;
}
#progreso{
  width: 0px;
  height: 16px;
  background: rgba(0,0,150,.2);
}
.clear{
  clear: both;
}
```

Código 6-12

Estilos CSS para el reproductor.

El documento CSS del Código 6-12 utiliza técnicas del modelo de caja tradicional para crear una caja que contiene todas las partes de un reproductor de vídeo. La caja está centrada en la ventana utilizando este modelo. Observe que también añade un al final del elemento <nav> para recuperar el flujo normal del documento.

No hay nada sorprendente en el último código CSS. Simplemente se trata de un grupo de propiedades (que ya hemos estudiado) que proporcionan estilos básicos a los elementos. Sin embargo, hay una característica que podría ser considerada inusual: la propiedad width para el elemento identificado como progreso,se ha inicializado en 0. Esto se debe a que este elemento se utiliza para simular una barra de progreso que crece a medida que el vídeo se reproduce.

6.4.2 Aplicación

Hay diferentes maneras de programar un reproductor de vídeo. En este libro presentaremos la mayor parte de los componentes de esta API pero solo veremos cómo utilizar los métodos y las propiedades elementales para el procesamiento de vídeo. Trabajaremos con algunas sencillas funciones para reproducir el vídeo, hacer una pausa, controlar el volumen, mostrar una barra de progreso mientras el vídeo se reproduce y dar la opción de hacer clic en esta barra para desplazarse hacia adelante y hacia atrás en la línea de tiempo.

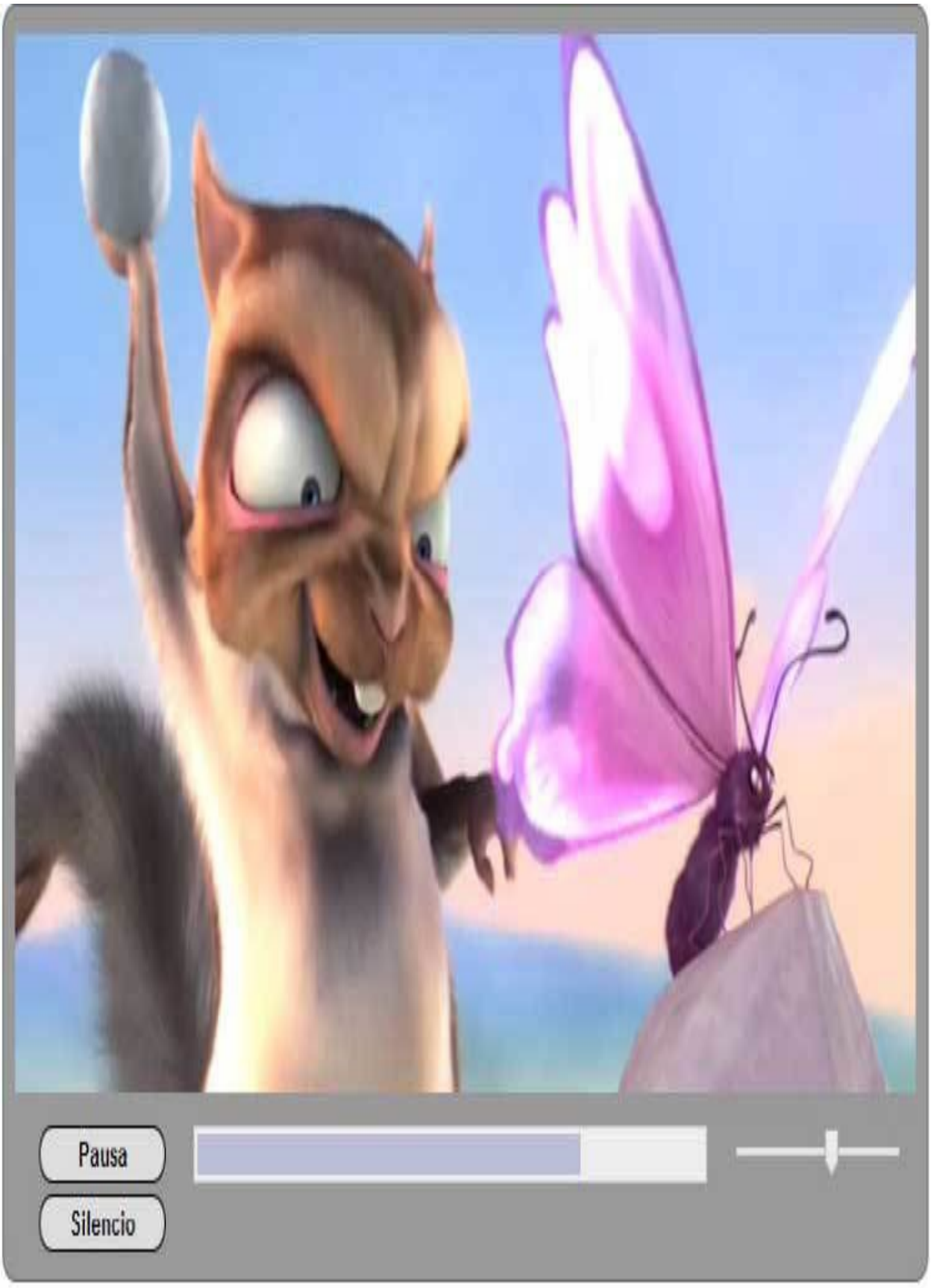

Pausa
Silencio

Figura 6-1

El reproductor de vídeo en funcionamiento. © Copyright 2008, Blender Foundation www.bigbuckbunny.org.

Hágalo usted mismo

Copie el Código 6-11 en un archivo vacío (por ejemplo, video.html). Cree dos nuevos archivos vacíos: uno para los estilos CSS y otro para los códigos Javascript. Estos deberán llamarse respectivamente reproductor.css y reproductor.js. Copie el Código 6-12 en el archivo CSS y luego copie todos los códigos Javascript que se enumeran a continuación en el archivo de Javascript.

6.4.3 Eventos

HTML5 incorpora nuevos eventos que son específicos para API. Para el procesamiento de vídeo y audio, se incorporaron eventos que informan a los usuarios acerca de la situación actual del material multimedia, por ejemplo, el progreso de la descarga del archivo, si el vídeo ha llegado al final, o si está en pausa o en proceso de reproducción, entre otros. No los vamos a utilizar en nuestro ejemplo, pero son necesarios para construir aplicaciones más complejas. Estos son los más comunes:

progress: Este evento se activa periódicamente para proporcionar actualizaciones sobre el progreso de la descarga de los medios de comunicación. Se puede acceder a la información a través del atributo buffered, como veremos más adelante.

canplaythrough: Este evento se activa cuando el archivo multimedia puede ser reproducido sin interrupciones. El estado se establece considerando la tasa de descarga actual y suponiendo que será la misma para el resto del proceso. Hay otro evento para este propósito, canplay, pero no tiene en cuenta la situación en su conjunto y se dispara cuando solo hay un par de cuadros disponible. Véase el ejemplo del Código 11-8, Capítulo 11).

ended: Este evento se activa cuando el material llega al final.

pause: Este evento se activa cuando el material está en pausa.

play: Este evento se activa cuando el material multimedia empieza a reproducirse.

error: Este evento se activa cuando se produce un error. Se envía al elemento <source> que corresponde a la fuente multimedia que produce el error.

6.4.4 Código Javascript

Para comenzar el código javascript definiremos algunas variables y añadiremos algunos detectores para eventos. Declararemos las variables en el espacio global para poder acceder a ellas desde cualquier parte del código. Las variables globales son buenas para las pruebas y propósitos didácticos, pero recuerde que debe evitarlas en aplicaciones reales y seguir la práctica común de declarar un objeto global y luego crear su aplicación dentro de este objeto (ver el ejemplo del Capítulo 10, Código 10-30).

```
var maxim, mmedia, play, bar, progress, mute, volume, loop;
function iniciar(){
  maxim = 400;
  mmedia = document.getElementById('media');
  play = document.getElementById('play');
  bar = document.getElementById('barra');
  progress = document.getElementById('progreso');
  mute = document.getElementById('mute');
  volume = document.getElementById('volume');
  play.addEventListener('click', push);
  mute.addEventListener('click', sound);
  bar.addEventListener('click', move);
  volume.addEventListener('change', level);
}
```

Código 6-13

Inicialización de la aplicación.

El Código 6-13 presenta la primera función para nuestro reproductor de vídeo. La función ejecuta la aplicación una vez que se carga el documento. En esta función se crea una referencia a cada elemento del reproductor con el selector getElementById y también establece la variable maxim para establecer el tamaño máximo de la barra de progreso (400 px).

Hay que tomar en cuenta varias acciones: cuando el usuario hace clic en los botones Reproducir y Silencio, cambia el volumen desde la entrada range o hace clic en la barra de progreso para avanzar o retroceder en la línea de tiempo. Por eso hemos añadido detectores del evento click a los elementos play, mute y barra y otro para la entrada range, que controla el volumen. Cada vez que el usuario hace clic en uno de estos elementos o mueve el control deslizante, las funciones de detector correspondientes se ejecutan: push() para el botón Reproducir, sound() para el botón Silencio, move() para la barra de progreso, y level() para la entrada volume.

6.4.5 Métodos

La función push() incorporada en el Código 6-14 será la primera en ejecutar una acción. Ejecutará los métodos play() y pause() de acuerdo con la situación.

```
function push(){
  if(!mmedia.paused && !mmedia.ended) {
    mmedia.pause();
    play.value = 'Reproducir';
    clearInterval(loop);
  }else{
    mmedia.play();
    play.value = 'Pausa';
    loop = setInterval(status, 1000);
  }
}
```

Código 6-14

Reproducir y detener el vídeo.

Los métodos play() y pause() son algunos de los métodos incorporados por HTML5 para el procesamiento de materiales multimedia. Estos son las más comunes:

play(): Reproduce el archivo multimedia.

pause(): Hace una pausa en el archivo multimedia.

load(): Este método carga el archivo multimedia. Es útil para cargar el material por adelantado para aplicaciones dinámicas.

canPlayType(type): Este método permite saber si un formato de archivo es compatible con el navegador o no. El atributo type es un tipo MIME para los archivos multimedia, como video/mp4 o video/ogg. El método puede devolver tres valores dependiendo de la seguridad de que se podrán reproducir los archivos multimedia: una cadena de texto vacía (no compatible), la cadena maybe (tal vez) y la cadena probably (probablemente).

6.4.6 Propiedades

La función push() también utiliza algunas propiedades para devolver información sobre el material multimedia. Estas son las más comunes:

pause: Devuelve true si la reproducción se ha detenido o no se ha iniciado.

ended: Devuelve true si la reproducción ha terminado.

duration: Devuelve la duración en segundos del material multimedia.

currentTime: Devuelve o establece un valor que informa sobre el punto de reproducción del material o se establece una nueva posición para empezar a reproducirlo.

volume: Devuelve o establece el volumen de reproducción. Los valores posibles van desde 0,0 a 1,0.

muted: Devuelve o establece el estado del audio. Devuelve los valores true (silenciado) o false (no silenciado).

error: Esta propiedad devuelve el valor de error si se produce un error.

buffered: Ofrece información sobre el nivel de carga del archivo en la memoria intermedia. Dado que los usuarios pueden forzar al navegador a descargar los archivos multimedia desde diferentes posiciones en la línea de tiempo, la información devuelta por buffered es una matriz que contiene todas las partes que se han descargado de los archivos; no solo la que comienza desde el principio. Los elementos de la matriz son accesibles por los atributos end() y start(). Por ejemplo, el código buffered.start(0) devolverá el tiempo en el que el primer fragmento del material multimedia comienza, y buffered.end(0) devolverá el tiempo en el que termina.

6.4.7 Código en funcionamiento

Ahora que conoce todos los elementos involucrados en el procesamiento de vídeo, vamos a echar un vistazo a cómo funciona la función push(). La función se ejecuta cuando el usuario hace clic en el botón Reproducir. Este botón tiene dos propósitos: muestra el texto Reproducir para reproducir el video o Pausa para detenerlo, de acuerdo a las circunstancias. Por tanto, cuando el vídeo está en pausa o no se ha iniciado, al pulsar el botón se reproduce el vídeo, pero ocurrirá lo contrario si el vídeo está siendo reproducido.

Para lograrlo, el código detecta el estado del material multimedia revisando las propiedades paused y ended. Esto se hace por la declaración if que se encuentra en la primera línea de la función. Si el valor de mmedia.paused y mmedia.ended es false, significa que el vídeo se está reproduciendo. Por lo tanto se ejecuta el método pause() para detener el vídeo y el texto del botón cambia a Reproducir. Observe que se ha utilizado el operador ! para cada propiedad (operador lógico de negación) para lograr este propósito. Si las propiedades devuelven el valor false, el operador cambia el valor a true. La declaración if debe leerse así: “si el material multimedia no está en pausa y el material multimedia no se ha acabado de reproducir, entonces haz esto”. Si el vídeo está en pausa o ha terminado la reproducción, la condición es false y el método play() se ejecuta para iniciar o reanudar el vídeo. En este caso, también estamos realizando una importante acción que es iniciar la ejecución de la función status() cada segundo con setInterval().

```
function status(){
  if(!mmedia.ended){
    var size = parseInt(mmedia.currentTime * maxim / mmedia.
 duration);
    progress.style.width = size + 'px';
  }else{
    progress.style.width = '0px';
    play.innerHTML = 'Reproducir';

    clearInterval(loop);
  }
}
```

Código 6-15

Actualización de la barra de progreso.

La función status() del Código 6-15 se ejecuta cada segundo mientras el vídeo se reproduce. Aquí también hay una declaración if para comprobar el estado de vídeo. Si la propiedad ended devuelve false, se calcula la longitud en píxeles que debería tener la barra de reproducción y se ajusta el tamaño del que lo representa. Si el valor de la propiedad ended es true (es decir, el vídeo ha terminado), se reestablece el tamaño de la barra de progreso de nuevo a 0 píxeles, se cambia el texto del botón por Reproducir y se cancela el bucle con clearInterval(). Después de hacer esto, la función status() deja de ejecutarse.

Veamos cómo se realiza el cálculo del tamaño de la barra de progreso. Debido a que la función status() se ejecuta cada segundo mientras que el vídeo se está reproduciendo, el tiempo actual cambia constantemente. Por esta razón, la propiedad currentTime debe ser consultada en cada ciclo. También hay que usar el valor de la propiedad duration para la duración del vídeo, y la variable maxim para el tamaño máximo de la barra de progreso. Con estos tres valores, se calcula la longitud en píxeles de la barra que representará los segundos ya reproducidos. La fórmula current-time × maxim / total-duration transforma los segundos en pixeles para cambiar el tamaño del que representa la barra de progreso.

Conceptos básicos

El método parseInt() es un método nativo de Javascript que analiza una cadena de texto y devuelve un número entero. Por lo general se utiliza para asegurar que

se obtiene un número cuando la fuente de información no es confiable, puede tener caracteres ocultos, o simplemente queremos extraer un número de un texto. También hay otro método para números de coma flotante llamado parseFloat().

La función para manipular el evento click para el elemento reproducir (el botón Reproducir) ya ha sido creada, ahora es el momento de hacer lo mismo con la barra de progreso.

```
function move(e){
  if(!mmedia.paused && !mmedia.ended){
    var mouseX = e.pageX - bar.offsetLeft;
    var newtime = mouseX * mmedia.duration / maxin;
    mmedia.currentTime = newtime;
    progress.style.width = mouseX + 'px';
  }
}
```

Código 6-16

Reproducción desde la posición seleccionada por el usuario.

En la función iniciar() se ha añadido un detector para el evento click del elemento bar para comprobar cada vez que el usuario desea iniciar la reproducción del vídeo desde una nueva posición. Cuando se activa el evento, la función move() se ejecuta para manejar la situación. Encontrará esta función en el Código 6-16. Se inicia con una declaración if, igual que las funciones anteriores, pero esta vez el objetivo es llevar a cabo la acción solo cuando el vídeo se está reproduciendo. Si las propiedades pause y end son false, la reproducción del vídeo y el código tiene que ejecutarse.

Hágalo usted mismo

Copie todos los códigos Javascript presentados desde el Código 6-13 dentro del archivo reproductor.js. Abra el archivo video.html que contiene el Código 6-11 en el navegador y haga clic en el botón Reproducir. Pruebe la aplicación en distintos navegadores.

Varias cosas son necesarias para calcular el punto en el que el vídeo debe empezar a reproducirse: determinar la posición del puntero del ratón cuando se llevó a cabo el evento click, la distancia en píxeles desde esa posición hasta el comienzo de la barra de progreso y el número de segundos que esa distancia representa en la línea de tiempo.

En la función move del Código 6-16, se utiliza la referencia al evento (e) junto con la propiedad de evento pageX para capturar la posición exacta del puntero del ratón cuando el usuario pulsa en la barra de progreso. Esta propiedad devuelve un valor en píxeles respecto al borde de la página y no a la barra de progreso o la ventana.

Para calcular el número de píxeles que hay desde el principio de la barra de progreso hasta la posición del puntero del ratón, se resta el espacio entre el lado izquierdo de la página y el principio de la barra. Recuerde que la barra de progreso se encuentra dentro de una caja que está centrada en la pantalla. Por tanto, vamos a decir que la barra está situada 421 píxeles desde la parte izquierda de la página y el clic se realizó en el centro de la barra. Debido a que la barra tiene 400 píxeles de largo, el clic se realizó a 200 píxeles del borde de la barra. Sin embargo, la propiedad pageX no devolverá 200; devolverá el valor 621 (421 + 200). Para obtener la posición exacta donde se hizo el clic en la barra, hay que restar la distancia desde el lado izquierdo de la página hasta el comienzo de la barra (en este ejemplo, 421 píxeles). Esta distancia se puede calcular mediante la propiedad offsetLeft y con la fórmula e.pageX - bar.offsetLeft se obtiene la posición exacta del puntero del mouse con respecto a la barra. En este ejemplo, el resultado final será 621 − 421 = 200.

Una vez que tenemos este valor, hay que convertirlo en segundos. Utilizando la propiedad duration, la posición exacta del ratón puntero en la barra, y el tamaño máximo de la barra, se construye la fórmula mouseX × video.duration / maximo para obtener el valor y almacenarlo en la variable newtime. El resultado es el tiempo en segundos que representa en la la posición del puntero del ratón en la línea de tiempo.

Conceptos básicos

Las propiedades pageX y offsetLeft aplicadas en el último ejemplo son propiedades tradicionales de Javascript. La primera de ellas, junto con pageY, devuelve la posición del puntero del ratón con respecto al borde de la página, mientras offsetLeft, junto con offsetTop, devuelve la posición del elemento respecto a su contenedor.

Ahora, hay que reproducir el vídeo desde la nueva posición. La propiedad currentTime, como hemos mencionado antes, devuelve el tiempo actual de reproducción del vídeo, pero también mueve el vídeo a un tiempo específico si un nuevo valor es asignado a la misma. Al asignar el valor de la variable newtime a la propiedad currentTime, el vídeo se desplaza a la posición deseada.

El último paso es para cambiar el tamaño del elemento progress para reflejar la nueva ubicación en la pantalla. Usando el valor en la variable mouseX, se puede modificar el estilo width del elemento para hacerlo crecer hasta la posición exacta donde se hizo el clic.

Aún debemos introducir dos pequeñas funciones para controlar el volumen de reroducción del audio. La primera es la función sound(), incluida para responder al evento click del botón Silencio.

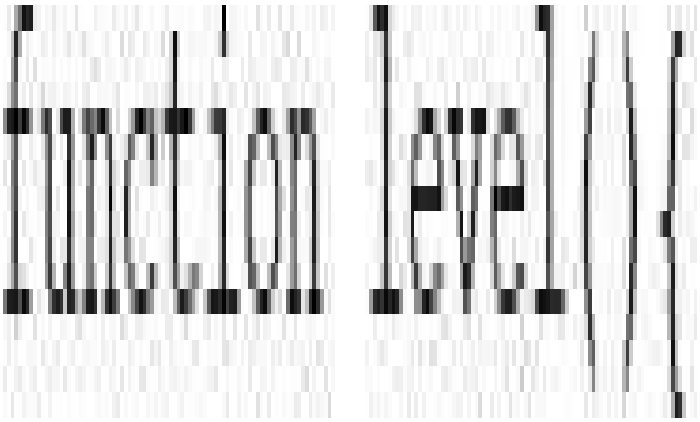
```
function sound(){
    if(mute.value == 'Silencio'){
        mmedia.muted = true;
        mute.value = 'Sonido';
    }else{
        mmedia.muted = false;
        mute.value = 'Silencio';
    }
}
```

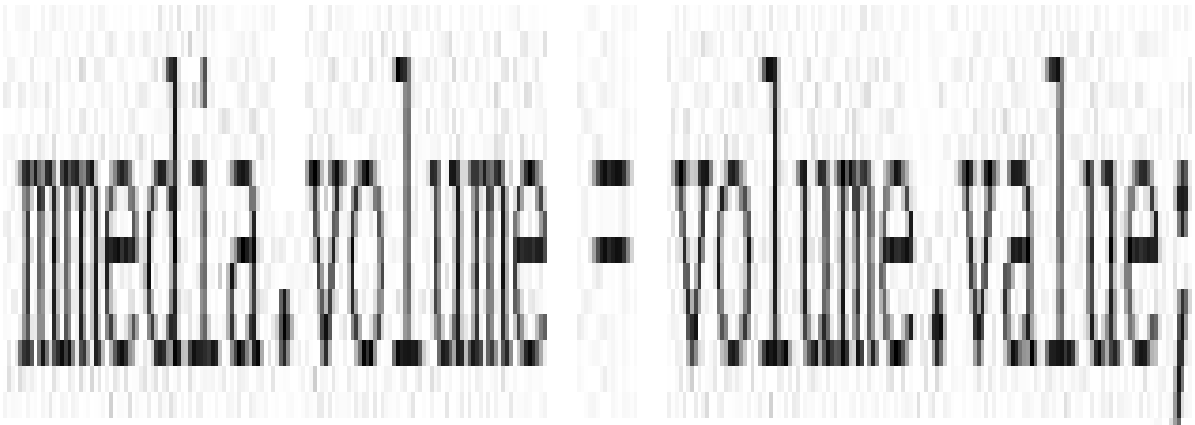
Código 6-17

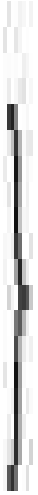
Activar y desactivar silencio con la propiedad muted.

La función del Código 6-17 activa o desactiva el audio del archivo multimedia en función del valor del atributo value del botón Silencio. El botón muestra diferentes textos de acuerdo con el estado. Si el valor actual es Silencio, el sonido se silencia y el valor del botón cambia a Sonido. De lo contrario, el audio se activa y el valor del botón cambia a Silencio.

Cuando el sonido está activo, el volumen puede ser controlado a través de la entrada range ubicada en el extremo de la barra de progreso. El elemento dispara el evento change cuando cambia su valor. Al evento lo maneja la función level().


function level1(){


media.volume = volume.value;

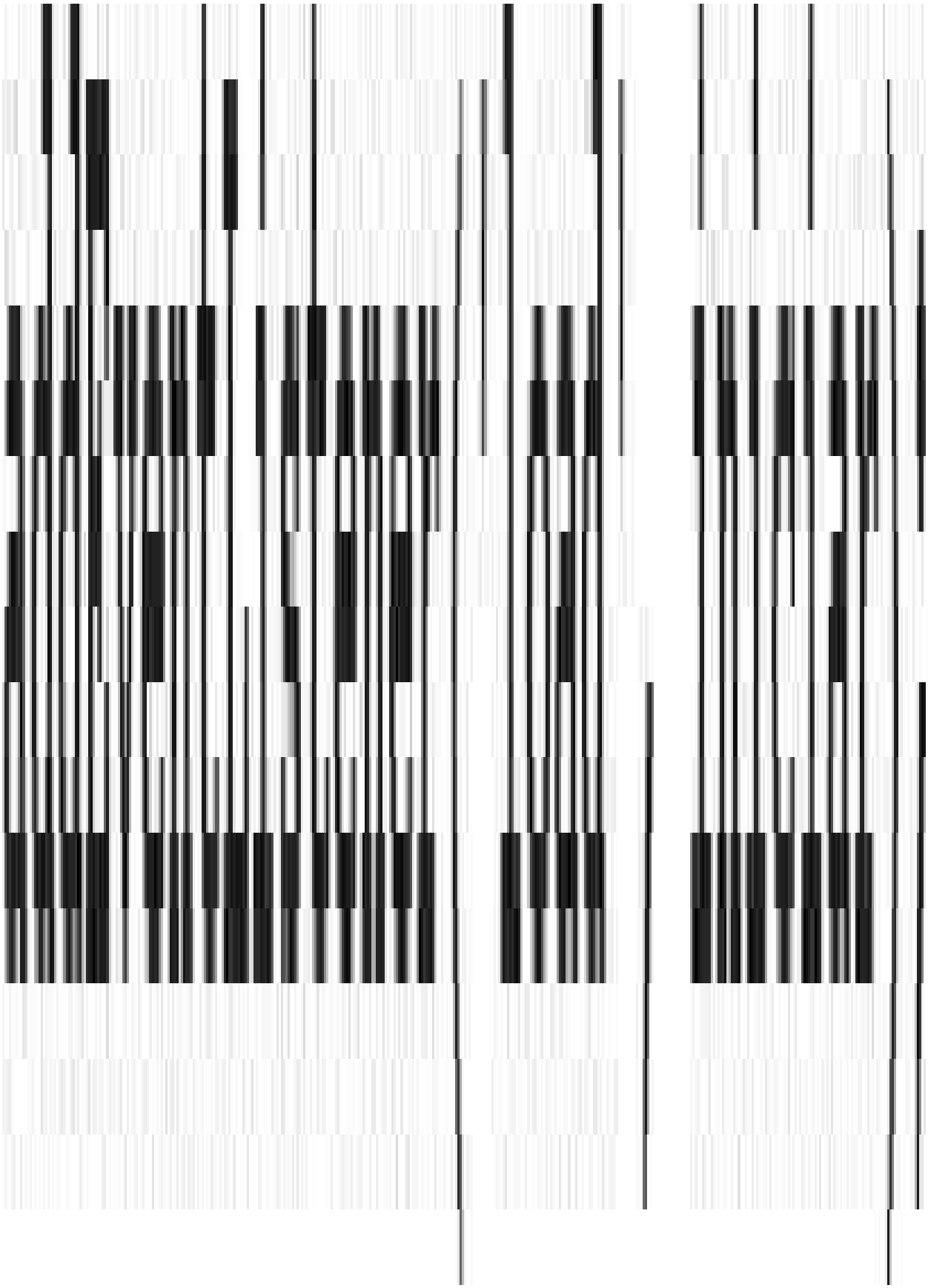


Código 6-18

Controlar el volumen.

La función del Código 6-18 simplemente asigna el valor del atributo value para el elemento <input> a la propiedad volume de los medios de comunicación. Lo único que tiene que tener en cuenta es que esta propiedad tiene valores digitales de 0,0 a 1. Los números fuera de este rango devolverán un error.

Con esta pequeña función, el código fuente para el reproductor de vídeo está casi listo: ya dispone de todos los eventos, métodos, propiedades y funciones que necesita para su aplicación. solo hay una línea más, un evento más que tiene que ser detectado para ejecutarlo.



Código 6-19

Detectar el evento para iniciar la aplicación.

7 API TextTrack

7.1 API TextTrack

Como mencionamos en el capítulo anterior, HTML5 introduce no solo el elemento <track> para subtítulos, sino también una nueva API que permite acceder al contenido de las pistas desde Javascript.

La API TextTrack proporciona un objeto llamado TextTrack para acceder al contenido de una pista. Hay dos maneras de llegar a este objeto: desde el contenido multimedia o desde el elemento <track>.

textTracks: Esta propiedad contiene una matriz con los objetos TextTrack correspondientes a cada pista del archivo multimedia. Los objetos TextTrack se almacenan en la matriz en orden secuencial, empezando desde el índice 0. La matriz, como cualquier otra, proporciona la propiedad length para obtener el número de objetos disponibles.

track: Esta propiedad devuelve el objeto TextTrack de la pista especificada.

Si el elemento multimedia (vídeo o audio) tiene varios elementos <track>, puede ser más fácil encontrar la pista deseada desde la matriz textTracks.

```
<!DOCTYPE html>
<html lang="es">
<head>
 <title>Trabajar con pistas</title>
 <script>
  function iniciar() {
   var video = document.getElementById('media');
   var track1 = video.textTracks[0];
   var mitrack = document.getElementById('mitrack');
   var track2 = mitrack.track;
   console.log(track1);
   console.log(track2);
  }
  addEventListener("load", iniciar);
 </script>
</head>
<body>
 <section>
  <video id="media" width="720" height="400" controls>
   <source src="trailer.mp4">
   <source src="trailer.ogg">
   <track id="mitrack" label="Subtítulos en inglés" src="subtitles.
 vtt" srclang="es" default>
  </video>
 </section>
</body>
</html>
```

Código 7-1

Conseguir el objeto TextTrack.

El documento del Código 7-1 muestra cómo obtener acceso al objeto TextTrack usando ambas propiedades. La función iniciar() primero crea una referencia al elemento <video> para obtener el objeto TextTrack de la pista y acceder a la matriz textTracks con el índice 0. Luego, se obtiene el mismo objeto del elemento <track> mediante la propiedad track. Finalmente, el contenido de ambas variables se imprime en la consola.

Hágalo usted mismo

Cree un nuevo archivo HTML con el documento del Código 7-1. Tiene que subir a su servidor todos los archivos a los que se hace referencia antes de realizar la prueba.

7.1.1 Lectura de pistas o tracks

Una vez obtenido el objeto TextTrack de la pista con la que se desea trabajar, estará en condiciones de acceder a sus propiedades:

kind: Devuelve el tipo de la pista, tal como es definido por el atributo kind del elemento <track>. Los valores posibles son: subtitles, captions, descriptions, chapters y metadata.

label: Devuelve la etiqueta de la pista, definida por el atributo label del elemento <track>.

language: Devuelve el idioma de la pista, según lo especificado por el atributo srclang del elemento <track>.

mode: Esta propiedad devuelve o establece el modo de la pista. Los tres valores posibles son disabled, hidden y showing. Se puede utilizar para cambiar las pistas.

cues: Es una matriz que contiene todas las entradas del archivo de la pista.

activeCues: Esta propiedad devuelve las entradas que se están visualizando en la pantalla (la anterior, la actual y la siguiente).

```
<!DOCTYPE html>
<html lang="es">
<head>
 <title>Trabajar con pistas</title>
 <style>
  #reproductor, #info{
   float: left;
  }
 </style>
 <script>
  function iniciar(){
   var lista = "";
   var info = document.getElementById('info');
   var miTrack = document.getElementById('miTrack');
   var obj = miTrack.track;

   lista += "<br>Tipo: " + obj.kind;
   lista += "<br>Etiqueta: " + obj.label;
   lista += "<br>Idioma: " + obj.language;
   info.innerHTML = lista;
  }

   addEventListener('load', iniciar);
 </script>
</head>
<body>
 <section id="reproductor">
  <video id="media" width="720" height="400" controls>
   <source src="trailer.mp4">
   <source src="trailer.ogg">
   <track id="miTrack" label="Subtítulos en inglés" src="subtitles.
 vtt" srclang="es" default>
  </video>
 </section>
 <aside id="info"></aside>
</body>
</html>
```

Código 7-2

Visualización de la información de la pista en la pantalla.

El documento de Código 7-2 incluye estilos para las dos columnas creadas por los elementos <section> y <aside> además de declaraciones en Javascript para obtener y mostrar los datos del elemento <track>. Como de costumbre, usamos el evento load para llamar a la función iniciar() tan pronto como se carga el documento. En esta función, primero definimos una variable vacía (lista) y obtenemos referencias para el elemento <aside> (info) y el elemento <track> (miTrack). Usando la última referencia, el objeto TextTrack es recuperado por la propiedad track y se almacena en la variable obj. De esta variable obtenemos las propiedades del objeto y generamos el texto que se mostrará en la pantalla. El texto se inserta finalmente en el elemento <aside> mediante la propiedad innerHTML.

Conceptos básicos

El operador += permite expresar de forma abreviada la siguiente operación: variable = variable + value, donde variable es la variable que utiliza el operador y value es el nuevo valor asignado a la variable. El operador solo añade el nuevo valor al valor actual de la variable.

7.1.2 Lectura de entradas o cues

Además de las aplicadas en el último ejemplo, hay una propiedad importante que llamada cues, que contiene una matriz con objeto TextTrackCues que representa cada entrada del archivo de pista.

```
function iniciar(){
 var lista = "";
 var info = document.getElementById('info');
 var miTrack = document.getElementById('miTrack');
 var obj = miTrack.track;
 var mispistas = obj.cues;
 for(var f = 0; f < mispistas.length; f++){
  lista += mispistas[f].text + "<br>";
 }
```

Código 7-3

Viendo las señales en la pantalla.

La nueva función iniciar() del Código 7-3 almacena la matriz cues en la variable mispistas y luego accede a cada entrada usando un bucle for. Debido a que la matriz se crea automáticamente, el primer índice es 0. Por esta razón, para leer cada una de las entradas, la variable f del bucle comienza en 0 (el valor inicial) y termina en el número de entradas (devuelto por la propiedad length de la matriz).

Los objetos TextTrackCue proporcionan propiedades para acceder a la información de la entrada. En el ejemplo se mostró el contenido de la propiedad text. A continuación presentamos una lista de las otras propiedades disponibles para estos objetos:

text: Devuelve el texto de la entrada.

startTime: Devuelve la hora de inicio de la entrada en segundos.

endTime: Devuelve la hora de finalización de la entrada en segundos.

vertical: Devuelve el valor del parámetro vertical. Si el parámetro no se define, el valor devuelto es una cadena vacía.

line: Devuelve el valor del parámetro line. Si el parámetro no se define, devuelve el valor predeterminado.

position: Devuelve el valor del parámetro position. Si el parámetro no se define, devuelve el valor predeterminado.

size: Devuelve el valor del parámetro size. Si el parámetro no se define, devuelve el valor predeterminado.

align: Devuelve el valor del parámetro align. Si el parámetro no se define, devuelve el valor predeterminado.

Hágalo usted mismo

Copie el Código 7-2 en un archivo HTML vacío y abra el archivo en su navegador. Para trabajar con entradas, sustituya la función iniciar() del Código 7-2 por la del Código 7-3.

7.1.3 Adición de pistas nuevas

El objeto TextTrack obtenido de las pistas también cuenta con tres importantes métodos para crear nuevas pistas y entradas desde Javascript:

addTextTrack(tipo, etiqueta, idioma): Este método crea una nueva pista para el medio especificado y devuelve el correspondiente objeto TextTrack. Es útil para crear nuevas pistas desde cero mediante programación. Los atributos son los valores de los atributos de la nueva pista y solo el tipo es obligatorio.

addCue(objeto): Añade una nueva entrada a la pista indicada. El atributo objeto es un objeto TextTrackCue devuelto por el constructor TextTrackCue().

removeCue(objeto): Este método elimina una entrada de la pista especificada. El atributo object es un objeto TextTrackCue devuelto por el objeto TextTrack.

Para añadir entradas a la pista, hay que proporcionar un objeto TextTrackCue. La API incluye un constructor para crear este objeto desde la información de la entrada:

TextTrackCue (horaInicio, horaFinal, texto): Este constructor devuelve un objeto TextTrackCue para utilizar con el método addCue(). Los atributos representan los datos para la entrada: hora de inicio, hora de finalización y

el texto de ésta.

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Trabajar con tracks</title>
<script>
  function iniciar(){
   var entrada;
   var entradas = [
    { start: 2.000, end: 7.000, text: 'Bienvenido'},
    { start: 10.000, end: 15.000, text: 'Esto es un ejemplo'},
    { start: 15.001, end: 20.000, text: 'de cómo añadir'},
    { start: 20.001, end: 25.000, text: 'una nueva pista.'},
    { start: 27.000, end: 30.000, text: '¡Hasta luego!'},
   ]
   var video = document.getElementById('media');
   var nuevaentrada = video.addTextTrack('subtitles');
   nuevaentrada.mode = 'showing';
   for(var f = 0; f < entradas.length; f++){
    entrada = new TextTrackCue(entradas[f].start, entradas[f].end,
                        entradas[f].text);
    nuevaentrada.addCue(entrada);
   }
   video.play();
  }
  addEventListener('load', iniciar);
 </script>
</head>
<body>
 <section>
  <video id="media" width="720" height="400" controls>
   <source src="trailer.mp4">
   <source src="trailer.ogg">
  </video>
 </section>
</body>
</html>
```

Código 7-4

Añadir pistas y entradas con Javascript.

Comenzamos la función iniciar() del Código 7-4 con la definición de la matriz entradas con cinco entradas o cues para nuestra nueva pista. Las entradas se declaran como elementos de la matriz. Cada entrada es un objeto con las propiedades start, end y text. Los valores de las horas de inicio y final no utilizan la sintaxis de un archivo WebVTT, sino que tienen que ser declarados en segundos como números decimales.

Las entradas se pueden añadir a una pista existente o a una nueva. En nuestro ejemplo hemos creado una nueva pista del tipo subtitles usando el método addTextTrack(). También hay que declarar la propiedad mode de esta pista con el valor showing, para ordenar al navegador que muestre la pista en la pantalla. Cuando la pista está lista, todas las entradas de la matriz entradas se convierten en objetos TextTrackCue y se añaden a la pista utilizando el método addCue(). Por último, el vídeo se reproduce usando el método play() estudiado en el Capítulo 6.

Hágalo usted mismo

Copie el documento del Código 7-4 en un archivo HTML vacío y abra el archivo en su navegador. Recuerde que debe subir los archivos a su servidor.

Importante

Para el momento de escribir este libro, los métodos y propiedades declarados por la especificación para seleccionar y activar pistas aún no han sido implementados en los navegadores. Para obtener más información y actualizaciones sobre este tema, por favor visite nuestro sitio web.

8 API Fullscreen

8.1 Basta de ventanas

Los desarrolladores tienen razón cuando dicen que la Web está cambiando. Se ha andado mucho camino desde aquellas páginas web enmarcadas con hipervínculos, iconos pequeños y feas barras de desplazamiento en todas partes. Ahora, la Web es una plataforma multipropósito y multimedia en la que todo es posible.

Desde los medios sociales hasta espacios de almacenamiento masivo, la Web está involucrada en cada aspecto de la vida de las personas; desde el trabajo hasta el entretenimiento, todas nuestras necesidades son atendidas.

Ante estos avances de la Web, el navegador se está convirtiendo en algo irrelevante, como una nueva forma de sistema operativo, está a cargo de todo pero escondido, sin casi notarse. Los menús se omiten y se han minimizado las opciones.

En la actualidad, las ventanas de los navegadores solo tienen una barra de navegación para indicar hacia dónde queremos ir. Con tantas nuevas aplicaciones absorbentes y la adopción de aplicaciones móviles independientes, la palabra navegación casi ha perdido importancia. La ventana del navegador no solo es innecesaria sino que a veces se puede conseguir en el camino. Por esta razón, HTML5 introduce la API Fullscreen.

8.1.1 Ir a pantalla completa

Fullscreen es una API de usos múltiples que permite ampliar cualquier elemento del documento de manera que ocupe toda la pantalla. Como resultado de esta nueva característica, la interfaz del navegador permanece oculta en el fondo y la atención del usuario se centra en vídeos, fotos, aplicaciones, videojuegos o lo que sea que esté viendo.

La API proporciona una serie de propiedades, métodos y eventos para hacer que un elemento se muestre a pantalla completa, salga del modo de pantalla completa o incluso para recuperar alguna información del elemento y del documento:

requestFullscreen(): Se aplica a todos los elementos del documento y activa el modo de pantalla completa para el mismo.

exitFullscreen(): Este método se aplica al documento. Si un elemento se encuentra en el modo de pantalla completa el método cancela el modo y devuelve el foco de nuevo a la ventana del navegador.

fullscreenElement: Esta propiedad devuelve una referencia al elemento que está en modo de pantalla completa. Si no hay ningún elemento en pantalla completa, la propiedad devuelve null.

fullscreenEnabled: Es una propiedad booleana que devuelve true cuando el documento es capaz de ir a pantalla completa o false en el caso contrario.

fullscreenchange: Este evento es disparado por el documento cuando un elemento entra o sale del modo de pantalla completa.

fullscreenerror: Este evento es lanzado por el elemento en caso de fallo (si el modo de pantalla completa no está disponible para ese elemento o para el documento). Este evento no devuelve datos.

El método requestFullscreen() y el evento fullscreenerror están asociados a los elementos del documento, pero el resto de las propiedades, métodos y eventos son parte del objeto Document, por lo tanto, son accesibles desde la propiedad document.

```
<!DOCTYPE html>
<html lang="es">
<head>
 <title>Pantalla completa</title>
 <script>
  var video;
  function iniciar(){
   video = document.getElementById('media');
   video.addEventListener('click', pantallacompleta);
  }
  function pantallacompleta(){
   if(!document.webkitfullscreenElement){
    video.webkitRequestFullscreen();
    video.play();
   }
  }
  addEventListener('load', iniciar);
 </script>
</head>
<body>
 <section>
  <video id="media" width="720" height="400">
   <source src="trailer.mp4">
   <source src="trailer.ogg">
  </video>
 </section>
</body>
</html>
```

Código 8-1

Llevar un elemento <video> a pantalla completa.

La función iniciar() del Código 8-1 añade un detector para el evento click del elemento <video>.

La función gofullscreen(), ajustada como detector para este evento, se ejecuta cada vez que el usuario hace clic en el vídeo. En esta función se utiliza la propiedad fullscreenElement para detectar si algún elemento se encuentra en modo de pantalla completa o no, y si no es así, el vídeo cambia a pantalla completa con el método requestFullscreen(). Al mismo tiempo, el vídeo se reproduce gracias a la aplicación del método play().

Importante

Ésta es una API experimental. Las propiedades, métodos y eventos requieren el uso de prefijos para los navegadores mientras la especificación final se implementa. Google Chrome y Mozilla Firefox son los navegadores que tienen actualmente una implementación en funcionamiento de la API Fullscreen. En el caso de Google Chrome, en lugar de los nombres originales se deben utilizar los siguientes:

webkitRequestFullscreen()

webkitExitFullscreen()

webkitfullscreenchange

webkitfullscreenerror

webkitFullscreenElement

Para Mozilla Firefox, los nombres son:

mozRequestFullScreen()

mozCancelFull-Screen()

mozfullscreenchange

mozfullscreenerror

mozFullScreenElement

Como estos nombres podrían haber cambiado después de la publicación de este libro, por favor visite nuestro sitio web para obtener actualizaciones.

8.1.2 Estilos “Fullscreen”

Los navegadores ajustan las dimensiones del elemento <video> al tamaño de la pantalla de forma automática, pero para cualquier otro elemento se conservan las dimensiones originales y el espacio libre en la pantalla se llena con un fondo negro. Por esta razón, la especificación proporciona una nueva pseudo-clase llamada :full-screen para modificar los estilos del elemento cuando se expande a pantalla completa.

```
<!DOCTYPE html>
<html lang="es">
<head>
 <title>Pantalla completa</title>
 <style>
  #reproductor:-webkit-full-screen, #reproductor:-webkit-full-screen
 #medios{
   width: 100%;
   height: 100%;
  }
 </style>
 <script>
  var video, reproductor;
  function iniciar(){
   video = document.getElementById('medios');
   reproductor = document.getElementById('reproductor');
   reproductor.addEventListener('click', pantallacompleta);
  }
  function pantallacompleta(){
   if(!document.webkitFullscreenElement){
    reproductor.webkitRequestFullscreen();
    video.play();
   }else{
    document.webkitExitFullscreen();
    video.pause();
   }
  }
  addEventListener('load', iniciar);
 </script>
</head>
<body>
 <section id="reproductor">
  <video id="medios" width="720" height="400">
   <source src="http://minkbooks.com/content/trailer.mp4">
   <source src="http://minkbooks.com/content/trailer.ogg">
  </video>
 </section>
</body>
</html>
```

Código 8-2

Tomar cualquier elemento de pantalla completa.

El Código 8-2 cambia al elemento <section> y su contenido a pantalla completa. La función pantallacompleta() ha sido modificada para poder activar y desactivar el modo de pantalla completa para este elemento. Como en el ejemplo anterior, el vídeo se reproduce cuando se encuentra en modo de pantalla completa, pero ahora se detiene cuando el modo se cancela.

Estas mejoras son suficientes para hacer realidad nuestro reproductor a pantalla completa. En el modo de pantalla completa, el nuevo contenedor para los elementos es la pantalla, pero las dimensiones originales y los estilos de los elementos <section> y <video> se conservan intactas. Utilizando la pseudo-clase :fullscreen se cambian los valores de las propiedades width y height de estos elementos a 100%, para igualar las dimensiones del contenedor. Ahora los elementos ocupan toda la pantalla y verdaderamente reflejan el modo de pantalla completa buscado.

Hágalo usted mismo

Cree un nuevo archivo HTML para probar los ejemplos en este capítulo. Una vez que el documento se abra en el explorador, haga clic en el vídeo para activar el modo de pantalla completa.

Importante

La última especificación declara el nombre de esta pseudo-clase sin el guión (:fullscreen), pero para el momento en el que estamos escribiendo este libro, navegadores como Mozilla Firefox y Google Chrome han implementado solo la especificación anterior y trabajan con el nombre mencionado en este capítulo (:full-screen). Por favor, visite nuestro sitio web para obtener actualizaciones.

9. API Stream

9.1 Capturar contenidos

La API Stream permite acceder a contenido multimedia de flujo continuo o en streaming proporcionados por el dispositivo. Las fuentes más comunes son la cámara web y el micrófono, pero la API permite acceder a cualquier otra fuente. La API especifica el objeto MediaStream para hacer referencia a los flujos continuos de medios y el objeto LocalMediaStream para los medios generados por los dispositivos locales. Estos objetos tienen una entrada representada por el dispositivo y una salida representada por el elemento <video>, el elemento <audio> y también otras API.

Para obtener el objeto LocalMediaStream, la API proporciona el método getUserMedia():

getUserMedia (límites, success, error): Este método pertenece al objeto navigator. Genera un objeto LocalMediaStream. El primer atributo es un objeto con dos propiedades booleanas vídeo o audio para indicar el tipo de contenido que debe ser capturado; el segundo atributo es una función que recibe y procesa el objeto LocalMediaStream, y el último atributo es otra función que procesa los errores.

El objeto LocalMediaStream hace referencia a los materiales multimedia, pero no es una fuente adecuada para los elementos de los medios de comunicación. Para convertir este objeto en una fuente apropiada para los elementos <video> y <audio>, tenemos que convertirlo en una URL. Javascript ofrece dos métodos generales para obtener una URL y trabajar con ella:

createObjectURL(datos): Este método pertenece al objeto URL. Devuelve una URL que podemos utilizar posteriormente para hacer referencia a los datos. El atributo datos puede ser un archivo, un objeto blob o, como en el caso de esta API, un objeto MediaStream.

revokeObjectURL(URL): Este método pertenece al objeto URL. Elimina una dirección URL creada por createObjectURL(). Es útil evitar el uso de viejas URL por scripts externos o, accidentalmente, desde nuestro propio código.

9.1.1 Acceder a la cámara web

Vamos a ver un ejemplo de cómo acceder a la cámara web y mostrar el vídeo en la pantalla.

```
<!DOCTYPE html>
<html lang="es">
<head>
 <title>API Stream</title>
 <script>
  function iniciar(){
   navigator.webkitGetUserMedia({video: true}, exito, mostrarerror);
  }
  function exito(stream){
   var video = document.getElementById("medio");
   video.setAttribute('src', URL.createObjectURL(stream));
   video.play();
  }
  function mostrarerror(e){
   console.log(e.code);
  }
  addEventListener('load', iniciar);
 </script>
</head>
<body>
 <section>
  <video id="medio"></video>
 </section>
</body>
</html>
```

Código 9-1

Acceder a la cámara web.

La función iniciar() del Código 9-1 obtiene el material multimedia de la cámara web con el método getUserMedia(). Ésta mostrará solicitud de autorización en la ventana del navegador. Si el usuario permite que la aplicación acceda a la cámara web, entonces se ejecuta la función exito(). Esta función recibe el objeto LocalMediaStream y lo almacena en la variable stream. Usando el método createObjectURL() se obtiene la URL que representa el stream o flujo de medios. La URL se asigna al atributo src del elemento <video>, y el vídeo se reproduce con el método play() estudiado en el Capítulo 6 (puede utilizarse en su lugar el atributo autoplay). Tenga en cuenta que en el ejemplo no ha hecho falta declarar el atributo src en la etiqueta <video> porque la fuente será el vídeo capturado por el código Javascript, pero se podrían haber establecido los atributos width y height para cambiar el tamaño del vídeo en la pantalla.

Importante

Los ejemplos de este capítulo utilizan el método con prefjijo de Google Chrome (sin ningún guión): webkitGetUserMedia(). Actualmente algunos navegadores como Opera tienen una implementación sin prefijos, pero otros no han implementado el método de ninguna forma. Le aconsejamos utilizar la última versión de Google Chrome para probar los códigos de este capítulo.

En caso de fallo, la API devuelve diferentes tipos de errores en función de la

aplicación. El más común es PERMISSION_DENIED (permiso denegado), producido cuando el usuario niega el acceso a los medios o los medios no están disponibles por otras razones. Cuando esto sucede, la propiedad error devuelve el valor 1.

Conceptos básicos

El objeto Navigator es un objeto tradicional que ofrece métodos y propiedades para acceder al navegador. solo lo utilizamos para aplicar los nuevos métodos incorporados por HTML5. Para obtener más información sobre los métodos y propiedades antiguos, visite nuestra página web y siga los enlaces de este capítulo.

9.1.2 Objetos MediaStreamTrack

Los objetos MediaStream (y por tanto los objetos LocalMediaStream) contienen objetos MediaStreamTrack que representan a cada pista multimedia (normalmente una para vídeo y otra para audio). Estos objetos se almacenan en matrices a las que se puede acceder por medio de las propiedades videoTracks y audioTracks, y proporcionan las propiedades siguientes:

enabled: Devuelve true o false según la condición de la pista. Si la pista está todavía asociada con la fuente, el valor es true.

kind: Esta propiedad devuelve el tipo de fuente de la pista representa. Tiene dos valores posibles: audio o video.

label: Esta propiedad devuelve el nombre de la fuente de la pista.

```
<!DOCTYPE html>
<html lang="es">
<head>
 <title>API Stream</title>
 <style>
  section{
   float: left;
  }
 </style>
 <script>
  function iniciar(){
   navigator.webkitGetUserMedia({video: true}, success, showerror);
  }
  function success(stream){
   var video = document.getElementById("video");
   video.setAttribute('src', URL.createObjectURL(stream));
   video.play();

   var datos = document.getElementById('datos');
   var track = stream.videoTracks[0];
   datos.innerHTML = '<br> Enabled: ' + track.enabled;
   datos.innerHTML += '<br> Type: ' + track.kind;
   datos.innerHTML += '<br> Device: ' + track.label;
  }
  function showerror(e){
   console.log(e.code);
  }
  addEventListener('load', iniciar);
 </script>
</head>
<body>
 <section>
  <video id="video"></video>
 </section>
 <section id="datos"></section>
</body>
</html>
```

Código 9-2

Mostrar información de la transmisión.

Las propiedades videoTracks y audioTracks son matrices. Para acceder a la pista generada por la cámara web, hay que leer la matriz videoTracks con el índice 0 (es decir, la primera pista de la lista). En el caso de que se hayan añadido varias pistas al objeto, puede consultarse la propiedad length para obtener el número total de pistas y crear un bucle for para acceder a cada una de ellas.

En el Código 9-2 se utiliza el mismo código del ejemplo anterior para acceder al stream, pero esta vez se almacena en la variable track la pista de la cámara web y luego se muestran en pantalla cada una de sus propiedades.

Los objetos MediaStreamTrack también ofrecen eventos para informar sobre el estado de la pista:

muted: Se activa cuando la pista no está en condiciones de proporcionar datos.

unmuted: Se activa tan pronto como la pista reinicia el suministro de datos.

ended: Se activa cuando la pista ya no puede proporcionar datos. Puede haber varias razones para ello, desde que el usuario ha negado el acceso a la cámara hasta el uso del método stop(), estudiado de inmediato.

Importante

Las pistas que se mencionan aquí son de vídeo o de audio. Este tipo de pistas no tienen nada que ver con las pistas de subtítulos opcionales estudiados en el Capítulo 6 y el Capítulo 7.

Estos eventos han sido diseñados para controlar la transmisión remota, un proceso estudiado en el Capítulo 24.

9.1.3 Método stop()

El objeto LocalMediaStream tiene el método stop() para detener manualmente la transmisión. He aquí un ejemplo:

```
<!DOCTYPE html>
<html lang="es">
<head>
 <title>Stream API</title>
 <script>
  function iniciar(){
   navigator.webkitGetUserMedia({video: true}, exito, mostrarerror);
  }
  function exito(stream){
   var botón= document.getElementById("botón");
   botón.addEventListener('click', function(){ stopstream(stream) });
   var video = document.getElementById("video");
   video.setAttribute('src', URL.createObjectURL(stream));
   video.play();
  }
  function mostrarerror(e){
   console.log(e.code);
  }
  function stopstream(stream){
   stream.stop();
   alert('Transmisión cancelada');
  }
  addEventListener('load', iniciar);
 </script>
</head>
<body>
 <section>
  <video id="video"></video>
 </section>
 <nav>
  <input type="botón" id="botón" value="Detener">
 </nav>
</body>
</html>
```

Código 9-3

Cancelación de la corriente.

En el Código 9-3 se incluye un botón para apagar la cámara. La función stopstream() maneja el evento clic de este botón. En esta función, el método stop() es llamado para detener la transmisión (la cámara es apagada) y el mensaje Transmisión cancelada se muestra en la pantalla.

Hágalo usted mismo

Trate de aplicar la propiedad CSS filter al elemento <video> de cualquiera de los ejemplos anteriores con los métodos estudiados en el Capítulo 3. Obtendrá interesantes efectos (por ejemplo, filter: grayscale(1);).

Conceptos básicos

Las funciones declaradas como atributos para el método getUserMedia(), success() y showError(), son funciones de devolución de llamada. En Javascript, las funciones de devolución de llamada son ampliamente utilizadas para procesar el resultado producido por la función primaria o método. En este caso, la función success() recibe un objeto que representa la transmisión de vídeo, y la función showError() recibe un evento con propiedades cuyos valores corresponden con

el error generado por el proceso. Las funciones de devolución de llamada están presentes en casi todas las API de Javascript. No se preocupe si no entiende este concepto de inmediato, después de trabajar más con este tipo de patrón, le resultará familiar.

10 API Canvas

10.1 Los gráficos para la Web

Al principio del libro explicamos que HTML5 está reemplazando anteriores plug-ins como Flash o Java applets, por ejemplo. A la hora independizar la red de tecnologías desarrolladas por terceros, había por lo menos dos cosas importantes que tomar en cuenta: el procesamiento de vídeo y los gráficos. El elemento <video> y las API estudiadas en capítulos anteriores cubren el primer aspecto muy bien, pero no se ocupan del segundo: los gráficos. Para este propósito, HTML5 introduce la API Canvas. Esta API maneja el aspecto gráfico y lo hace de una manera extraordinariamente eficiente. Permite dibujar, representar gráficos, animar y procesar imágenes y texto, y trabaja junto con el resto de las API para crear aplicaciones completas e incluso videojuegos en 2D y 3D para la Web.

10.1.1 El elemento <canvas>

El elemento <canvas> (lienzo) genera un espacio vacío de forma rectangular, en el que se mostrarán los resultados de los métodos proporcionados por la API. Produce un espacio en blanco, como un elemento vacío, pero para un fin completamente diferente.

Importante

Para fines de compatibilidad, en caso de que la API Canvas no esté disponible en el navegador, el contenido que se encuentre entre las etiquetas <canvas> será mostrado en pantalla.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>API Canvas</title>
  <script src="canvas.js"></script>
</head>
<body>
  <section id="cajadelienzo">
    <canvas id="lienzo" width="500" height="300"></canvas>
  </section>
</body>
</html>
```

Código 10-1

Uso del elemento <canvas>.

Solo necesitamos especificar unos pocos atributos para crear este elemento, como se puede ver en el Código 10-1. Los atributos width y height declaran el tamaño de la caja y son necesarios porque todo lo que sea dibujado sobre el lienzo tendrá estos valores como referencia. Éste es básicamente el propósito del elemento <canvas>: crea una caja vacía en la pantalla y, gracias a Javascript y los métodos y propiedades introducidos por la API, la superficie se convierte en algo práctico.

10.1.2 GetContext()

El método getContext() es el primero que tenemos que llamar para preparar al elemento <canvas> para trabajar. Genera un contexto de dibujo que se asigna al lienzo. Desde la referencia que devuelve será posible aplicar el resto de la API.

```
function iniciar(){

 var elem = document.getElementById('lienzo');

 lienzo = elem.getContext('2d');

}

addEventListener("load", iniciar);
```

Código 10-2

Crear el contexto de dibujo para el lienzo.

En el Código 10-2 se almacena una referencia al elemento <canvas> en la variable elem y se crea el contexto de dibujo con getContext(‘2d’).

Hágalo usted mismo

Copie el documento HTML del Código 10-1 en un nuevo archivo vacío. Además, cree un archivo denominado canvas.js y copie el Código 10-2 en este archivo. Cada ejemplo de este capítulo es independiente y sustituye al anterior.

El método puede tomar dos valores: 2d y webGL. Estos establecen un entorno bidimensional o uno tridimensional, respectivamente. Actualmente, 2d está disponible en todos los navegadores compatibles con HTML5, mientras que WebGL solo es aplicable en los navegadores que han implementado y habilitado el uso de biblioteca WebGL para la generación de gráficos 3D. Estudiaremos WebGL en el próximo capítulo.

El contexto de dibujo 2D del lienzo será una cuadrícula de píxeles en filas y columnas que van de arriba a abajo y de izquierda a derecha, con su origen (el píxel 0, 0) situado en la esquina superior izquierda del lienzo.

10.2 Dibujar en el lienzo

Una vez que el elemento <canvas> y su contexto están listos, podemos empezar a crear y manipular gráficos reales. La lista de las herramientas que proporciona la API para este propósito es muy amplia, lo que permite la generación de efectos múltiples, desde la creación de formas simples y dibujos hasta textos, sombras o transformaciones complejas. En esta sección del capítulo estudiaremos estos métodos uno por uno.

10.2.1 Dibujar rectángulos

Por lo general, el desarrollador debe preparar la figura que se dibujará antes de enviarla al contexto (como veremos en breve), pero hay algunos métodos que permiten dibujar directamente en el lienzo. Estos métodos son específicos para las formas rectangulares y son los únicos que generan una forma primitiva (para obtener otras formas tendremos que combinar otras técnicas de dibujo y trazados complejos).

Los métodos disponibles son los siguientes:

fillRect(x, y, ancho, alto): Este método dibuja un rectángulo sólido. La esquina superior izquierda se encuentra en la posición especificada por los atributos x e y. Los atributos ancho y alto declaran el tamaño del rectángulo.

strokeRect(x, y, ancho, alto): Similar al método anterior pero éste dibuja un rectángulo vacío, es decir, solo el contorno.

clearRect(x, y, ancho, alto): Este método se usa para sustraer píxeles de la zona especificada por sus atributos. Es como un borrador rectangular.

```
function iniciar(){
  var elemento = document.getElementById('lienzo');
  var lienzo = elemento.getContext('2d');

  lienzo.strokeRect(100, 100, 120, 120);
  lienzo.fillRect(110, 110, 100, 100);
  lienzo.clearRect(120, 120, 80, 80);
}

addEventListener("load", iniciar);
```

Código 10-3

Dibujo de rectángulos.

Ésta es la misma función que muestra en el Código 10-2 pero incorpora algunos de los nuevos métodos estudiados para dibujar una figura en el lienzo. Como se puede ver, el contexto fue asignado a la variable lienzo, y luego esta variable fue utilizada para hacer referencia al contexto en cada método.

El primer método usado en la función, strokeRect(100, 100, 120, 120), dibuja un rectángulo vacío con la esquina superior izquierda en la posición 100, 100 y un tamaño de 120 píxeles. El segundo método, fillRect(110, 110, 100, 100), dibuja un rectángulo sólido, esta vez partiendo de la posición 110, 110 del lienzo. Y, finalmente, con el último método, clearRect(120, 120, 80, 80), es sustraído del centro de la figura un espacio cuadrado de 80 píxeles.

La Figura 10-1 es solo una representación de lo que verá después de la ejecución del Código 10-3.

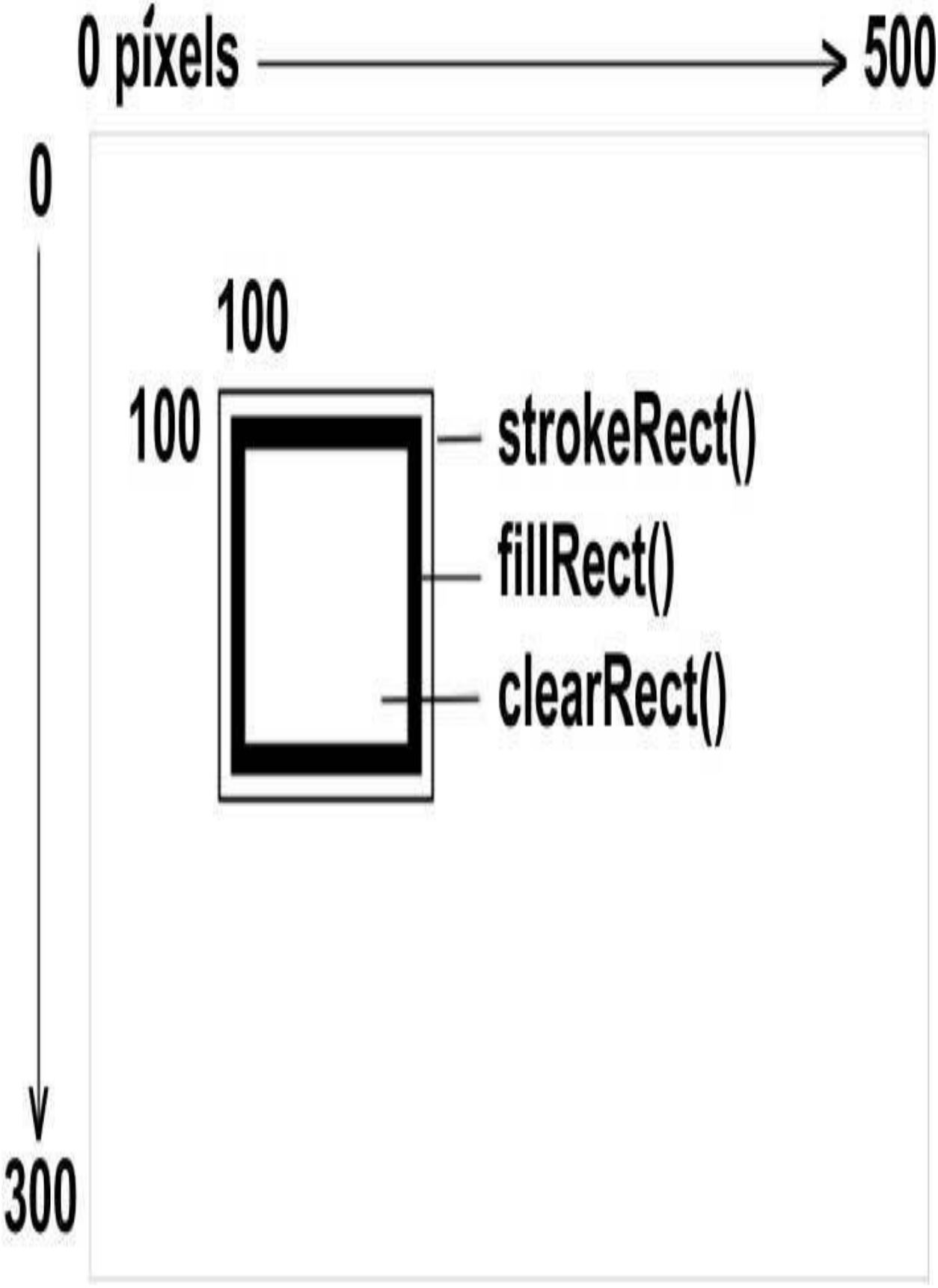

0 píxels
500
0
100
100
strokeRect()
fillRect()
clearRect()
300

Figura 10-1

Representación del lienzo y los rectángulos dibujados por el Código 10-3.

El elemento <canvas> se representa como una cuadrícula con su origen en la esquina superior izquierda y el tamaño especificado en sus atributos. Los rectángulos se dibujan en el lienzo en la posición declarada por los atributos x e y y, uno encima de otro de acuerdo al orden del código fuente (el primero en aparecer en el código fuente se dibuja, y el segundo dibuja sobre el primero, etc.). Existe un método para personalizar cómo se dibujan las formas en la pantalla, pero lo veremos más adelante.

10.2.2 Color

Hasta ahora hemos estado usando el color por defecto, negro, pero se puede especificar cualquier otro utilizando la sintaxis de CSS y las siguientes propiedades:

strokeStyle: Esta propiedad declara el color de las líneas de la forma.

fillStyle: Esta propiedad declara el color del interior de la forma.

globalAlpha: Esta propiedad no declara color sino la transparencia de todas las formas dibujadas en el lienzo.

```
function iniciar(){
  var elemento=document.getElementById('lienzo');
  lienzo=elemento.getContext('2d');

  lienzo.fillStyle="#000099";
  lienzo.strokeStyle="#990000";

  lienzo.strokeRect(100,100,120,120);
  lienzo.fillRect(110,110,100,100);
  lienzo.clearRect(120,120,80,80);
}
addEventListener("load", iniciar);
```

Código 10-4

Añadir color.

Los colores del Código 10-4 fueron declarados usando números hexadecimales. También pueden usarse funciones como rgb() o incluso especificar la transparencia para la forma mediante el aprovechamiento de la función rgba(). El valor de estos métodos siempre tiene que estar entre comillas, por ejemplo, strokeStyle = “rgba(255, 165, 0, 1)”.

Cuando se especifica un color mediante el uso de estos métodos, se convierte en el color predeterminado para el resto de los dibujos. Aunque es posible utilizar la función rgba(), también se puede utilizar otra propiedad para ajustar el nivel de transparencia. Hablamos de globalAlpha. Su sintaxis es globalAlpha=valor, donde valor es un número entre 0,0 (completamente opaco) y 1,0 (completamente transparente).

10.2.3 Degradados

Los degradados son una parte esencial de todo programa de dibujo de hoy y la API Canvas no es una excepción. Igual que en CSS3, los gradientes en el lienzo puede ser lineales o radiales, y podemos proporcionar puntos de parada para combinar colores:

createLinearGradient(x1, y1, x2, y2): Este método crea un objeto para aplicar un degradado lineal al lienzo.

createRadialGradient(x1, y1, r1, x2, y2, r2): Crea un objeto para aplicar un degradado radial al lienzo usando dos círculos. Los valores representan la posición del centro de cada círculo y sus radios.

addColorStop(posición, color): Este método especifica los colores que serán usados para el degradado. El atributo posición es un valor entre 0.0 y 1.0 que determina dónde comenzará el degradado para ese color en particular.

```
function iniciar (){

  var elemento = document.getElementById('lienzo');
  var lienzo = elemento.getContext('2d');

  var gradiente = lienzo.createLinearGradient(0,0,10,100);
  gradiente.addColorStop(0.5, '#0000FF');
  gradiente.addColorStop(1, '#000000');
  lienzo.fillStyle=gradiente;

  lienzo.fillRect(10, 10, 100, 100);
  lienzo.fillRect(150, 10, 200, 100);
}
addEventListener("load", iniciar);
```

Código 10-5

La aplicación de un degradado lineal al lienzo.

En el Código 10-5, creamos un objeto Gradient desde la posición 0,0 hasta la posición 10,100, otorgando una leve inclinación hacia la izquierda. Los colores son fijados por el método addColorStop(), y el gradiente se aplica a la propiedad fillStyle, como se haría con un color regular.

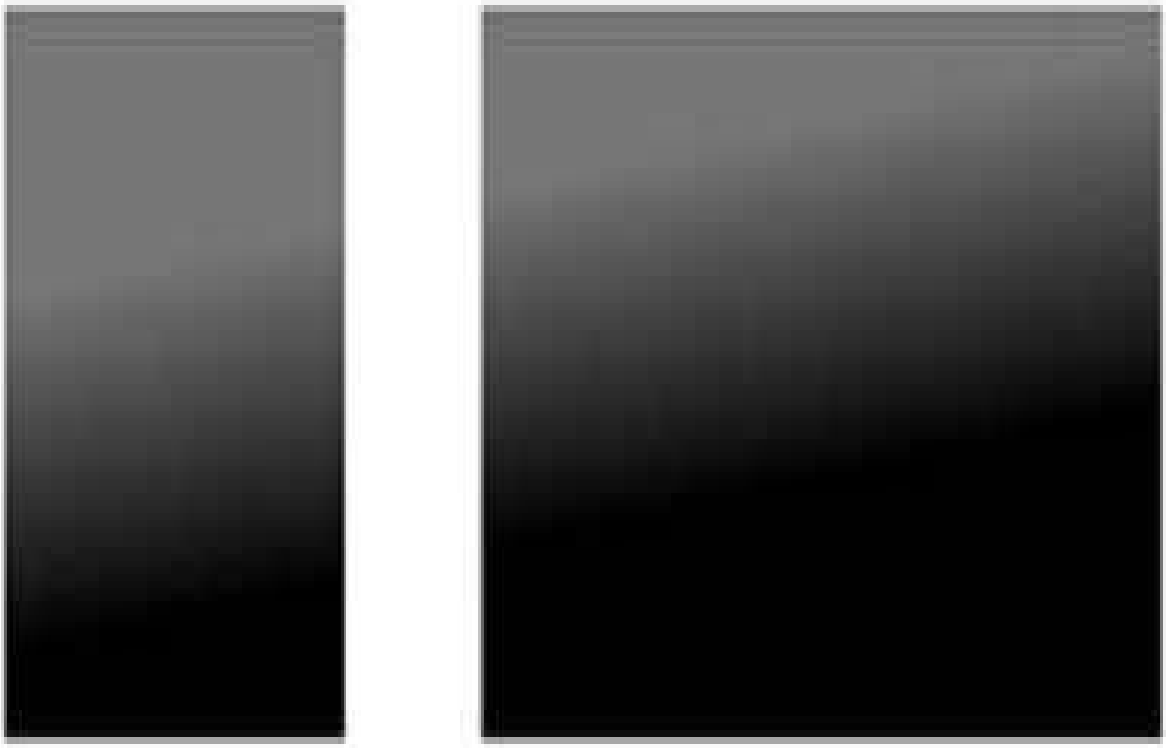


Figura 10-2

Degradado lineal para el lienzo.

Observe que las posiciones del degradado corresponden al lienzo, no a las figuras que queremos modificar. El resultado es que si movemos los rectángulos dibujados al final de la función hacia una nueva posición, cambiará el degradado para esos rectángulos.

Hágalo usted mismo

El degradado radial es similar al de CSS3. Intente reemplazar el degradado lineal del Código 10-5 por un gradiente radial usando una expresión como createRadialGradient(0, 0, 30, 0, 0, 300). También puede experimentar cambiando la posición de los rectángulos para ver cómo se aplica a estas figuras el degradado.

10.2.4 Crear trazados

Los métodos estudiados hasta el momento dibujan directamente en el lienzo, pero ese no es siempre el caso. Normalmente es necesario procesar figuras en segundo plano y, una vez que el trabajo está hecho, enviar el resultado al contexto para que sea dibujado. Con este propósito, la API Canvas introduce varios métodos con los que podremos generar trazados.

Un trazado es como el mapa que será seguido por un lápiz. Una vez declarado, el trazado es enviado al contexto y dibujado en el lienzo. El trazado puede incluir diferentes tipos de trazos, como líneas rectas, arcos y rectángulos, entre otros, para crear figuras complejas. Hay dos métodos para iniciar y cerrar una ruta:

beginPath(): Este método comienza la descripción de una nueva figura.

closePath(): Este método cierra el trazado generando una línea recta desde el último punto hasta el punto de origen. Puede ser ignorado si se se desea un trazado abierto o si se usa el método fill() para dibujar el trazado en el lienzo.

Por otra parte, también contamos con tres métodos para dibujar el trazado en el lienzo:

stroke(): El método dibuja el trazado como una forma vacía (solo el contorno).

fill(): Dibuja el trazado como una forma sólida. Cuando se utiliza este método, no es necesario cerrar el trazado con closePath(). El trazado se cierra automáticamente con una línea recta que va desde el punto final hasta el origen.

clip(): Declara una nueva área de corte para el contexto. Cuando el contexto es inicializado, el área de recorte es toda el área ocupada por el lienzo. Este método clip() cambia el área de recorte por una nueva forma y crea de este modo una máscara. Todo lo que esté fuera de esa máscara no será dibujado.

```
function iniciar(){
  var elemento = document.getElementById('lienzo');
  var lienzo=elemento.getContext('2d');


lienzo.beginPath();
// aquí va el trazado
lienzo.stroke();
}
addEventListener("load", iniciar);
```

Código 10-6

Inicio y final de un trazado.

Conceptos básicos

Los comentarios pueden ser insertados en el código Javascript usando dos barras para comentarios de una sola línea (// comentario) y la combinación de una barra y un caracter estrella para comentarios de varias líneas (/* comentario */).

El Código 7-6 no crea nada, solo incorpora los métodos necesarios para iniciar y luego dibujar el trazado en pantalla. Para definir el trazado y la forma, contamos con varios métodos:

moveTo(x, y): Este método mueve la pluma a una posición específica. Permite comenzar o continuar el trazado desde diferentes puntos de la cuadrícula, evitando las líneas continuas.

lineTo(x, y): Este método genera una línea recta desde la posición actual de la pluma hasta la nueva, declarada por los atributos x e y.

rect(x, y, ancho, alto): Este método genera un rectángulo. A diferencia de los métodos estudiados anteriormente, éste formará parte del trazado (no se

dibuja directamente en el lienzo). Los atributos tienen las mismas funciones que los métodos anteriores.

arc(x, y, radio, ánguloinicio, ángulofinal, dirección): Este método genera un arco o un círculo con el centro en las coordenadas de x e y, con el radio y los ángulos declarados por sus atributos. El último valor es un valor booleano (true o false) que indica la dirección a favor o en contra de las agujas del reloj.

quadraticCurveTo(cpx, cpy, x, y): Este método genera una curva Bézier cuadrática desde la posición actual de la pluma hasta la posición declarada por los atributos x y y. Los atributos cpx y cpy definen un punto x de control que dará forma a la curva.

bezierCurveTo(cp1x, cp1y, cp2x, cp2y, x, y): Este método es similar al anterior pero agrega dos atributos más para generar una curva de Bézier cúbica. Proporciona dos puntos de control para moldear la curva, declarados por los atributos cp1x, cp1y, cp2x y cp2y.

Veamos un trazado sencillo para entender cómo funcionan:

```
function iniciar(){
  var elemento = document.getElementById('lienzo');
  var lienzo = elemento.getContext('2d');
  lienzo.beginPath();
  lienzo.moveTo(100, 100);
  lienzo.lineTo(200, 200);
  lienzo.lineTo(100, 200);
  lienzo.stroke();
}
addEventListener("load", iniciar);
```

Código 10-7

Crear un trazado.

Recomendamos siempre establecer la posición inicial de la pluma inmediatamente después de iniciar el trazado con beginPath(). En el Código 7-7, el primer paso fue mover el lápiz a la posición 100, 100 y el siguiente generar una línea desde ese punto hasta el punto 200, 200. Así la posición del lápiz es 200, 200 y la siguiente línea es generada desde este punto hasta el punto 100, 200. Finalmente, el trazado es dibujado en el lienzo como una forma vacía con el método stroke().

La Figura 10-3 muestra una representación del triángulo abierto producido por el script del Código 10-7. Este triángulo puede ser cerrado o incluso rellenado utilizando diferentes métodos, como se muestra en los ejemplos siguientes:

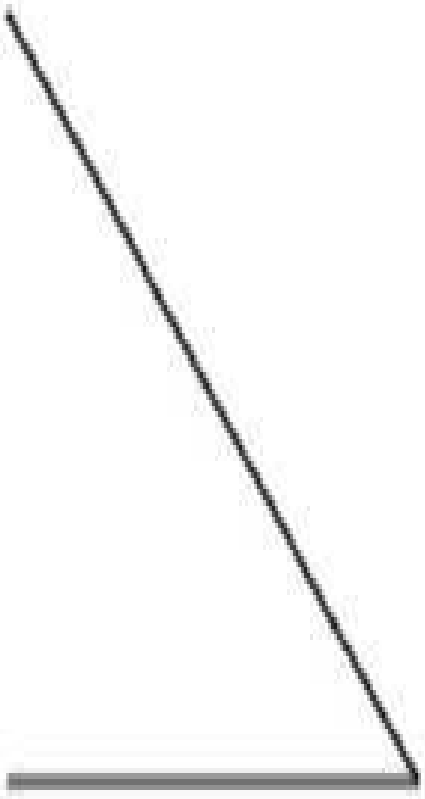


Figura 10-3

Trazado abierto.

```
function iniciar(){
  var elemento=document.getElementById('lienzo');
  var lienzo=elemento.getContext('2d');

  lienzo.beginPath();
  lienzo.moveTo(100,100);
  lienzo.lineTo(200,200);
  lienzo.lineTo(100,200);
  lienzo.closePath();
  lienzo.stroke();
}
addEventListener("load", iniciar);
```

Código 10-8

Completar el triángulo.

El método closepath() simplemente añade una línea recta al trazado, desde el punto final hasta el punto de inicio para cerrar la forma.

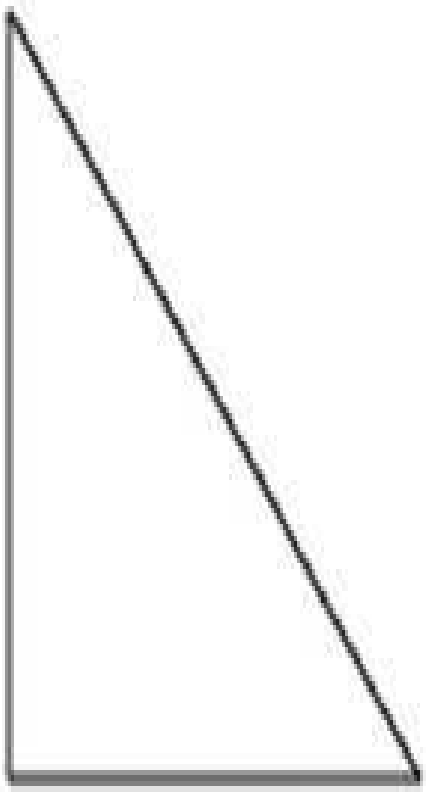


Figura 10-4

Trazado cerrado.

Usando el método stroke() al final del trazado dibujamos un triángulo vacío en el lienzo. Para lograr una figura sólida, este método debe ser reemplazado por el método fill():

```
function iniciar(){
  var elemento = document.getElementById('lienzo');
  var lienzo = elemento.getContext('2d');
  lienzo.beginPath();
  lienzo.moveTo(100,100);
  lienzo.lineTo(200,200);
  lienzo.lineTo(100,200);
  lienzo.fill();
}
addEventListener("load", iniciar);
```

Código 10-9

Dibujo de un triángulo sólido.

Ahora, la figura de la pantalla es un triángulo sólido. El método fill() cierra la ruta de forma automática, por lo que ya no es necesario utilizar closepath().

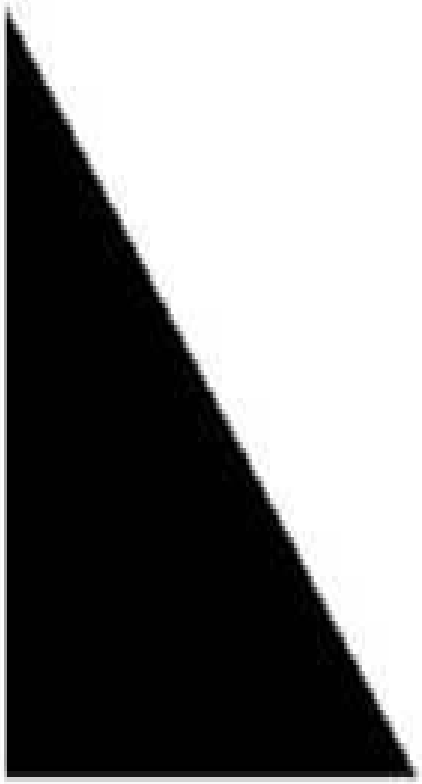


Figura 10-5

Trazado cerrado y triángulo sólido.

Anteriormente presentamos el método clip() para dibujar un trazado en el lienzo. Este método en realidad no dibuja nada, lo que hace es crear una máscara con la forma del trazado para seleccionar qué será dibujado y qué no. Todo lo que queda fuera de la máscara no se dibuja.

```
function iniciar(){
  var elemento =document.getElementById('lienzo');
  var lienzo = elemento.getContext('2d');
  lienzo.beginPath();
  lienzo.moveTo(100,100);
  lienzo.lineTo(200,200);
  lienzo.lineTo(100,200);
  lienzo.clip();
   lienzo.beginPath();
  for(f=0; f<300; f=f+10){
   lienzo.moveTo(0,f);
   lienzo.lineTo(500,f);
  }
  lienzo.stroke();
}
addEventListener("load", iniciar);
```

Código 10-10

Usar el triángulo como una máscara.

Para mostrar exactamente cómo funciona el método clip(), en el Código 7-10 utilizamos un bucle for para crear líneas horizontales cada 10 píxeles. Estas líneas van desde el lado izquierdo al lado derecho del lienzo, pero solo serán dibujadas las partes de las líneas que caen dentro de la máscara (el triángulo).

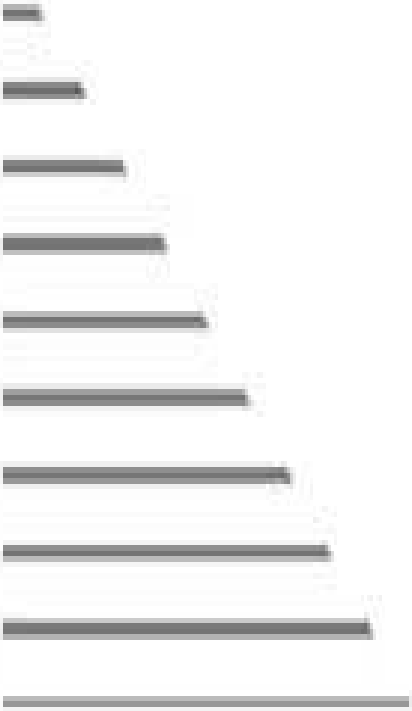


Figura 10-6

Máscara triangular.

Ahora que ya sabemos cómo dibujar trazados, es tiempo de ver el resto de las alternativas con las que contamos para crear formas. Hasta el momento hemos estudiado cómo generar líneas rectas y formas rectangulares. Para figuras circulares, la API provee tres métodos: arc(), quadraticCurveTo() y bezierCurveTo(). El primero es relativamente sencillo y puede generar círculos parciales o completos, como mostramos en el siguiente ejemplo:

```
function iniciar(){
  var elemento = document.getElementById('lienzo');
  var lienzo = elemento.getContext('2d');

  lienzo.beginPath();
  lienzo.arc(100, 100, 50, 0, Math.PI * 2, false);
  lienzo.stroke();
}
addEventListener("load", iniciar);
```

Código 10-11

Dibujar círculos con arc().

Lo primero que seguramente notará en el método arc() de nuestro ejemplo es el uso del valor PI. Este método usa radianes en lugar de grados para los valores del ángulo. En radianes, el valor PI representa 180 grados, por lo que la fórmula PI*2 multiplica PI por 2 obteniendo un ángulo de 360 grados.

El Código 7-11 genera un arco con centro en el punto 100, 100 y un radio de 50 píxeles, comenzando a 0 grados y terminando a Math.PI x 2°, lo que representa un círculo completo. El uso de la propiedad PI del objeto Math nos permite obtener el valor preciso de PI.

Conceptos básicos

Math es un objeto nativo de Javascript con propiedades y métodos para realizar operaciones matemáticas. Las propiedades devuelven constantes como PI o E, mientras que los métodos realizan operaciones complejas tales como la exponenciación o la trigonometría. Para obtener una lista completa, por favor visite nuestro sitio web y siga los enlaces disponibles para este capítulo.

Si necesita calcular el valor en radianes de cualquier ángulo en grados, use la

fórmula: Math.PI / 180 * grados, como en el próximo ejemplo:

```
function iniciar(){
 var elemento = document.getElementById('lienzo');
 var lienzo = elemento.getContext('2d');


 lienzo.beginPath();
 var radianes=Math.PI/180*45;
 lienzo.arc(100, 100, 50, 0, radianes, false);
 lienzo.stroke();
}
addEventListener("load", iniciar);
```

Código 7-12

Dibujar un arco de 45 grados.

Con el guión en el Código 10-12, se obtiene un arco que abarca 45 grados de un círculo. Cambiar el valor de la dirección a true (verdadero) y compruebe cómo en este caso, el arco será generado de 0 a 315 grados, creando un círculo abierto:

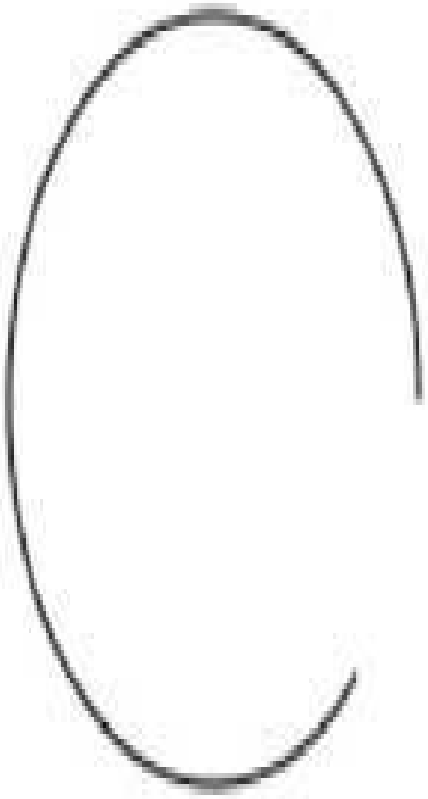


Figura 10-7

Círculo abierto creado con el método arc().

Una cosa importante a considerar es que si continúa trabajando con este trazado, el próximo punto de inicio será el final del arco. Si no desea partir de este punto tendrá que usar el método moveTo() para cambiar la posición de la pluma, como hicimos anteriormente. Sin embargo, si la próxima frma es otro arco (por ejemplo, un círculo completo), recuerde que el método moveTo() mueve la pluma virtual hacia el punto en el cual el círculo comenzará a ser dibujado, no el centro del círculo. Digamos que el centro del círculo que queremos dibujar se encuentra en el punto 300, 150 y su radio es de 50. El método moveTo() debería mover el lápiz a la posición 350, 150 para comenzar a dibujar el círculo.

Además de arc(), existen dos métodos más para dibujar curvas, en este caso curvas complejas. El método quadraticCurveTo() genera una curva Bézier cuadrática y el método bezierCurveTo() genera curvas Bézier cúbicas. La diferencia entre estos dos métodos es que el primero cuenta con un solo punto de control y el segundo con dos, creando de este modo diferentes tipos de curvas.

```
function iniciar(){
  var elemento = document.getElementById('lienzo');
  lienzo = elemento.getContext('2d');
  lienzo.beginPath();
  lienzo.moveTo(50,50);
  lienzo.quadraticCurveTo(100, 125, 50,200);
  lienzo.moveTo(250, 50);
  lienzo.bezierCurveTo(200, 125, 300, 125, 250, 200);
  lienzo.stroke();
}
addEventListener("load", iniciar);
```

Código 10-13

Creación de curvas complejas.

Para la curva cuadrática movimos el lápiz virtual a la posición 50, 50 y finalizamos la curva en el punto 50, 200. El punto de control para esta curva está en la posición 100, 125. La curva generada por el método bezierCurveTo() es un poco más compleja. Tiene dos puntos de control, el primero en la posición 200,125 y el segundo en la posición 300,125.

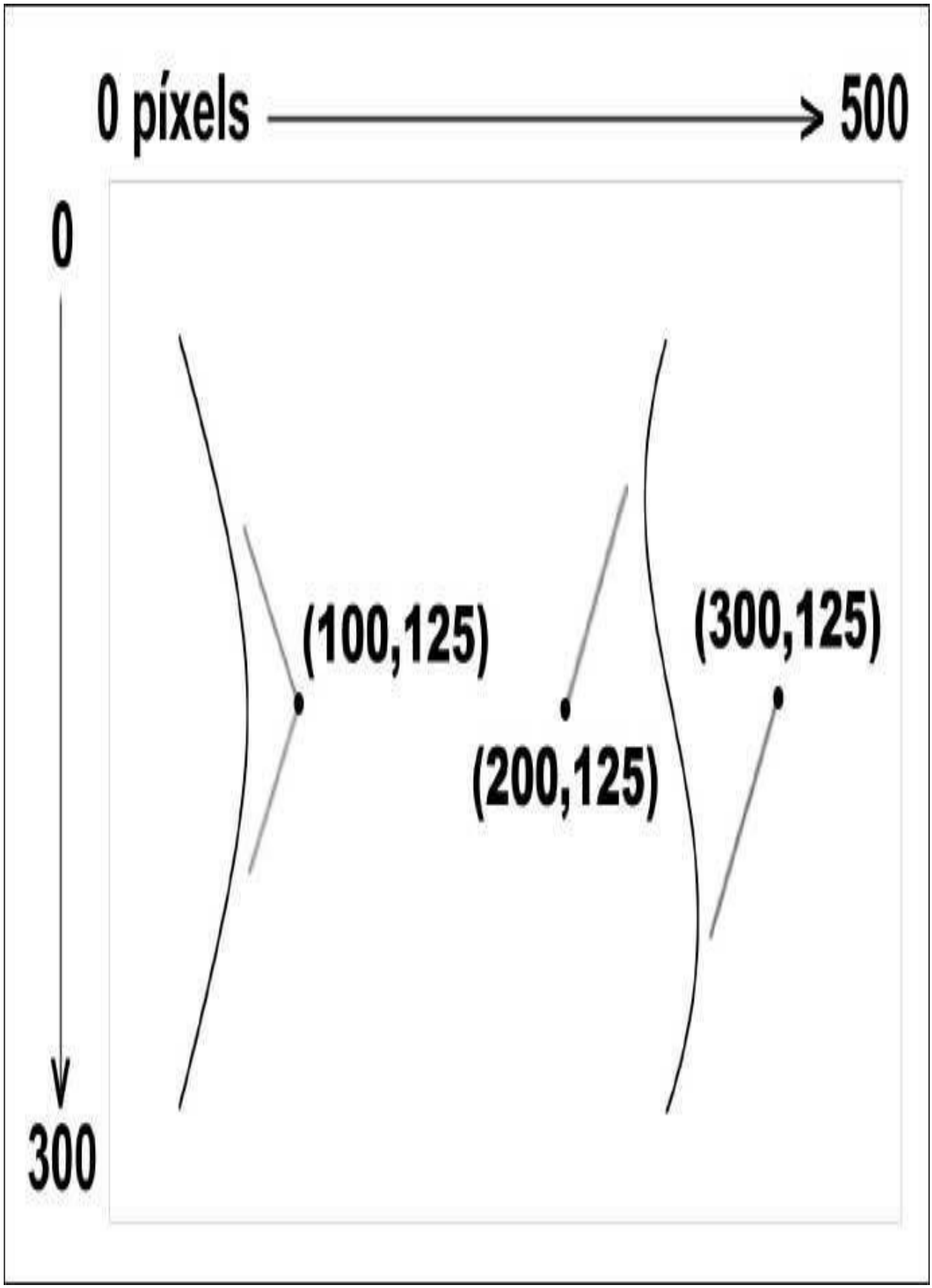

0 píxels
500
0
300
(100,125)
(200,125)
(300,125)

Figura 10-8

Representación de las curvas de Bézier y sus puntos de control en el lienzo.

Los valores en la Figura 7-2 indican los puntos de control de las curvas. Moviendo estos puntos se cambia la forma de la curva.

Hágalo usted mismo

Puede agregar tantas curvas como necesite para construir su figura. Intente cambiar los valores de los puntos de control del Código 7-13 para comprobar cómo afectan a las curvas. Construya figuras más complejas combinando curvas y líneas para entender cómo está construido el trazado.

10.2.5 Estilos de línea

De momento hemos usado siempre los mismos estilos de línea pero el ancho, la terminación y otros aspectos de la línea pueden ser modificados para obtener exactamente el tipo de línea que necesitamos para nuestros dibujos. Existen cuatro propiedades específicas para este propósito, que estudiaremos a continuación.

lineWidth: Esta propiedad determina el grosor de la línea. Por defecto, el valor es de 1,0 unidades.

lineCap: Esta propiedad determina la forma del extremo de la línea. Hay tres posibles valores: butt, round o square.

lineJoin: Esta propiedad determina la forma de la conexión entre dos líneas. Los valores posibles son round, bevel o miter.

miterLimit: Trabaja junto con lineJoin para determinar hasta qué punto se amplían las conexiones entre líneas cuando la propiedad lineJoin se declara con el valor miter.

Estas propiedades afectan a todo el trazado. Cada vez que hay que cambiar las características de las líneas, es necesario crear un nuevo trazado con nuevos valores de propiedades.

```
function iniciar(){
  var elemento = document.getElementById('lienzo');
  lienzo = elemento.getContext('2d');

  lienzo.beginPath();
  lienzo.arc(200, 150, 50, 0, Math.PI*2, false);
  lienzo.stroke();
  lienzo.lineWidth=10;
  lienzo.lineCap = "round";
  lienzo.beginPath();
  lienzo.moveTo(230, 150);
  lienzo.arc(200, 150, 30, 0, Math.PI, false);
  lienzo.stroke();

  lienzo.lineWidth = 5;
  lienzo.lineJoin = "miter";
  lienzo.beginPath();
  lienzo.moveTo(195,135);
  lienzo.lineTo(215,155);
  lienzo.lineTo(195,155);
  lienzo.stroke();
}
addEventListener("load", iniciar);
```

Código 10-14

Probar las propiedades de las líneas.

Comenzamos el dibujo en el ejemplo del Código 10-14 creando una ruta para un círculo completo con propiedades predeterminadas. Luego, utilizando lineWith, cambiamos el ancho de la línea a 10 y definimos la propiedad lineCap como round. Esto hace que el siguiente trazado sea más grueso con terminaciones redondeadas, lo que nos ayudará a crear una boca sonriente. Para crear la ruta, pasamos la pluma a la posición 230, 150 y luego generamos un semicírculo.

Finalmente, añadir una ruta creada por dos líneas para formar una figura como una nariz. Observe que las líneas de este camino tiene una anchura de 5 y se unen con la propiedad lineJoin establece en el valor miter. Esta propiedad hace que la nariz sea puntiaguda, expandiendo los bordes exteriores de la esquina hasta llegar a un solo punto.

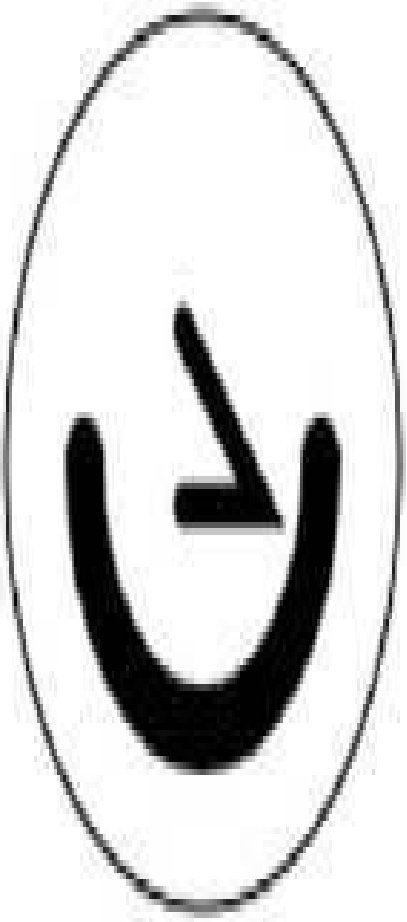

ر

Figura 10-9

Diferentes tipos de líneas.

Hágalo usted mismo

Experimente con las líneas de la nariz modificando la propiedad miterLimit (por ejemplo, con la instrucción miterLimit=2). Cambie el valor de la propiedad lineJoin a round o bevel. También puede modificar la forma de la boca probando diferentes valores para la propiedad lineCap ¡y acabar la cara añadiendo un par de ojos!

10.2.6 Texto

Escribir texto en el lienzo es tan simple como definir algunas propiedades y llamar al método apropiado. Hay tres propiedades para configurar el texto:

font: Esta propiedad establece el tipo de letra a utilizar y tiene una sintaxis similar los mismos valores que la propiedad font de CSS.

textAlign: Alínea el texto horizontalmente. Existen varios valores posibles: start, end, left, right y center.

textBaseline: Define la alineación vertical. Establece diferentes posiciones para el texto (incluyendo texto Unicode) y los valores posibles son: top, hanging, middle, alphabetic, ideographic o bottom.

Por otra parte, hay dos métodos disponibles para dibujar texto sobre el lienzo. Veamos cuáles son:

strokeText(texto, x, y): De forma similar que el método stroke() para el trazado, este método dibuja en la posición x,y el texto especificado como un contorno de forma. Puede también incluir un cuarto valor para declarar el tamaño máximo. Si el texto es más extenso que este último valor, será reducido para que quepa dentro del espacio establecido.

fillText(texto, x, y): Este método es similar al método anterior excepto que dibuja un texto sólido.

```
function iniciar(){
  var elemento = document.getElementById('lienzo');

  var lienzo = elemento.getContext('2d');


  lienzo.font= "bold 24px verdana, sans-serif";
  lienzo.textAlign= "start";
  lienzo.fillText("Mi mensaje", 100,100);
}
addEventListener("load", iniciar);
```

Código 10-15

Dibujar texto.

Como puede ver en el Código 7-15, la propiedad font puede tomar varios valores a la vez y usa exactamente la misma sintaxis que en CSS. La propiedad textAling hace que el texto sea dibujado desde la posición 100,100 (si el valor de esta propiedad fuera end, por ejemplo, el texto terminaría en la posición 100,100). Finalmente, el método fillText dibuja un texto sólido en el lienzo.

Importante

El método measureText() devuelve un objeto con varias propiedades. La propiedad width utilizada en este ejemplo es solo una de ellas. Para obtener una lista completa, consulte la especificación de la API Canvas. El enlace está disponible en www.minkbooks.com.

Además de los ya mencionados, la API proporciona otro método importante para trabajar con el texto:

measureText(text): Este método retorna información sobre el tamaño de un texto específico. Puede ser útil para combinar texto con otras formas en el lienzo y calcular posiciones o incluso colisiones en animaciones.

```
function iniciar(){
  var elemento=document.getElementById('lienzo');
  var lienzo=elemento.getContext('2d');

  lienzo.font="bold 24px verdana, sans-serif";
  lienzo.textAlign="start";
  lienzo.textBaseline="bottom";
  lienzo.fillText("Mi mensaje", 100,124);

  var tamano=lienzo.measureText("Mi mensaje");
  lienzo.strokeRect(100,100,tamano.width,24);
}
addEventListener("load", iniciar);
```

Código 10-16

Medir texto.

En este ejemplo se parte de la misma secuencia de comandos del Código 10-15, pero se añade una alineación vertical. La propiedad TextBaseline se establece con el valor bottom, lo que significa que la parte inferior del texto está en la posición 124. Esto ayuda a conocer la posición vertical exacta del texto en el lienzo.

Usando el método measureText() y su propiedad width se obtiene el tamaño horizontal del texto. Con todas las medidas tomadas, ahora es posible dibujar un rectángulo que rodee el texto.

Mi mensaje

Figura 10-10

Trabajar con texto.

Hágalo usted mismo

Use el texto del Código 10-16, pruebe diferentes valores para las propiedades textAlign y textBaseline. Utilice el rectángulo como una referencia para comprobar cómo funcionan. Escriba un texto diferente para ver cómo el rectángulo se adapta automáticamente a su tamaño.

10.2.7 Sombras

Las sombras son, por supuesto, también una parte importante de la API Canvas. Podemos generar sombras para todos los trazados e incluso para los textos. La API proporciona cuatro propiedades para hacelo:

shadowColor: Declara el color de la sombra usando sintaxis CSS.

shadowOffsetX: Recibe un número para determinar qué tan lejos del objeto, en dirección horizontal, estará ubicada la sombra.

shadowOffsetY: Recibe un número para determinar qué tan lejos del objeto, en dirección vertical, estará ubicada la sombra.

shadowBlur Esta propiedad aplica un efecto de desenfoque a la sombra.

```
function iniciar(){
  var elemento=document.getElementById('lienzo');
  var lienzo=elemento.getContext('2d');

  lienzo.shadowColor="rgba(0,0,0,0.5)";
  lienzo.shadowOffsetX=4;
  lienzo.shadowOffsetY=4;
  lienzo.shadowBlur=5;

  lienzo.font="bold 50px verdana, sans-serif";
  lienzo.fillText("Mi mensaje ", 100,100);
}
addEventListener("load", iniciar);
```

Código 10-17

Aplicar sombras.

Hágalo usted mismo

Aplique sombras a una figura en lugar de texto. Por ejemplo, pruebe generar sombras para figuras vacías y sólidas, usando rectángulos o círculos.

La sombra creada en el Código 10-17 utiliza la función rgba() para obtener un color negro semitransparente. Se desplaza 4 píxeles desde el objeto y tiene un valor de desenfoque de 5.

10.2.8 Transformaciones

Canvas permite realizar operaciones complejas en los gráficos y en el propio lienzo. Estas operaciones se realizan utilizando cinco métodos de transformación diferentes, cada uno con un propósito específico.

translate(x, y): Este método de transformación es usado para mover el origen del lienzo. Cada lienzo comienza en el punto 0,0 localizado en la esquina superior izquierda y los valores se incrementan en cualquier dirección dentro del lienzo. Los valores negativos caen fuera del lienzo. A veces es útil poder usar valores negativos para crear figuras complejas. El método translate() permite mover el punto 0,0 a una posición específica para usar el origen como referencia.

rotate(ángulo): Rota el lienzo alrededor del origen tantos radianes como sean indicados.

scale(x, y): Incrementa o disminuye las unidades de la cuadrícula del lienzo para reducir o ampliar todo lo que esté dibujado en él. La escala puede ser cambiada de forma independiente para los valores horizontal y vertical usando los atributos x e y. Los valores pueden ser negativos, produciendo un efecto de espejo. Por defecto los valores son iguales a 1, 0.

transform(m1, m2, m3, m4, dx, dy): El lienzo contiene una matriz de valores que especifican sus propiedades. El método transform() aplica una nueva matriz sobre la actual para modificar el lienzo.

setTransform(m1, m2, m3, m4, dx, dy): Reinicializa la matriz de transformación y establece una nueva desde los valores provistos por sus atributos.

```
function iniciar(){
  var elemento = document.getElementById('lienzo');
  var lienzo = elemento.getContext('2d');

  lienzo.font="bold 20px verdana, sans-serif";
  lienzo.fillText("PRUEBA",50,20);

  lienzo.translate(50,70);
  lienzo.rotate(Math.PI/180*45);
  lienzo.fillText("PRUEBA",0,0);

  lienzo.rotate(-Math.PI/180*45);
  lienzo.translate(0,100);
  lienzo.scale(2,2);
  lienzo.fillText("PRUEBA",0,0);
  }
window.addEventListener("load", iniciar);
```

Código 10-18

Trasladar, rotar y escalar.

No hay mejor forma de entender cómo funcionan las transformaciones que usarlas en nuestro código.

En el Código 10-18 aplicamos los métodos translate(), rotate() y scale() al mismo texto. Primero dibujamos un texto en el lienzo con la configuración por defecto. El texto aparecerá en la posición 50, 20 con un tamaño de 20 px. Luego de esto, usando translate(), el origen del lienzo es movido a la posición 50, 70 y el lienzo completo es rotado 45 grados con el método rotate(). Otro texto es dibujado en el nuevo origen, con una inclinación de 45 grados. Las transformaciones aplicadas se vuelven valores por defecto, por tanto antes de aplicar el siguiente método scale() rotamos el lienzo 45 grados negativos para ubicarlo en su posición original y además movemos el origen otros 100 píxeles hacia abajo. Finalmente, la escala del lienzo es duplicada y es dibujado un nuevo texto con el doble de tamaño del texto original.

PRUEBA

PRUEBA

PRUEBA

Figura 10-11

Aplicar transformaciones.

Cada transformación es acumulativa. Si realizamos dos transformaciones usando scale(), por ejemplo, el segundo método realizará el escalado considerando el estado actual del lienzo. Una orden scale(2, 2), luego de una orden scale(2, 2), cuadruplicará la escala del lienzo. Los métodos de transformación de la matriz no son una excepción y contamos con dos métodos para realizar esta clase de transformaciones: transform() y setTransform().

```
function iniciar(){
  var elemento = document.getElementById('lienzo');
  var lienzo = elemento.getContext('2d');

  lienzo.transform(3, 0, 0, 1, 0, 0);
  lienzo.font="bold 20px verdana, sans-serif";
  lienzo.fillText("PRUEBA",20,20);
  lienzo.transform(1, 0, 0, 10, 0, 0);
  lienzo.font="bold 20px verdana, sans-serif";
  lienzo.fillText("PRUEBA", 20, 20);
}
addEventListener("load", iniciar);
```

Código 10-19

Transformaciones acumulativas sobre la matriz.

Igual que en el código anterior, en el Código 10-19 se aplican varios métodos de transformación sobre el mismo texto para comparar sus efectos. Los valores por defecto de la matriz del lienzo son 1, 0, 0, 1, 0, 0. Cambiando el primer valor a 3, en la primera transformación de nuestro ejemplo anterior, estiramos el lienzo horizontalmente. El texto dibujado luego de esta transformación es más ancho de lo que sería en condiciones por defecto. Con la siguiente transformación, el lienzo es estirado verticalmente cambiando el cuarto valor a 10 y preservando el resto como estaba.

Un detalle importante a recordar es que las transformaciones son aplicadas sobre la matriz declarada en transformaciones previas, por lo que el segundo texto mostrado por el Código 10-19 será estirado horizontal y verticalmente tal como se muestra en la Figura 10-12. Para restablecer la matriz y declarar nuevos valores de transformación, podemos usar el método setTransform().

PRUEBA

PRUEBA

Figura 10-12

Modificar la matriz de transformación.

Hágalo usted mismo

Reemplace el último método transform() del documento de ejemplo por setTransform() y compruebe los resultados. Usando solo un texto, cambie cada valor en el método transform() para comprobar la clase de transformación realizada en el lienzo por cada uno de ellos.

10.2.9 Restaurar el estado

La acumulación de transformaciones hace realmente difícil volver a anteriores estados. En el Código 10-18, por ejemplo, tuvimos que recordar el valor de rotación usado previamente para poder realizar una nueva rotación que devolviera el lienzo al estado original. Considerando situaciones similares, API Canvas proporciona dos métodos para guardar y recuperar el estado del lienzo:

save(): Graba el estado del lienzo, incluyendo transformaciones ya aplicadas, valores de las propiedades de estilo y el trazado actual de recorte (el área creada por el método clip(), si la hay).

restore(): Este método recupera el último estado grabado.

```
function iniciar(){
var elemento=document.getElementById('lienzo');
var lienzo=elemento.getContext('2d');

lienzo.save();
lienzo.translate(50, 70);
lienzo.font="bold 20px verdana, sans-serif";
lienzo.fillText("PRUEBA1", 0, 30);
lienzo.restore();
lienzo.fillText("PRUEBA2", 0, 30);
}
addEventListener("load", iniciar);
```

Código 10-20.

Grabar el estado del lienzo.

Si ejecuta las órdenes Javascript del Código 10-20 en su navegador, verá el texto PRUEBA1 en grandes letras en el centro del lienzo y el texto PRUEBA2 en letras pequeñas, cerca del origen. Lo que hace éste código es grabar el estado por defecto del lienzo y luego establecer una nueva posición para el origen además de estilos para el texto. Antes de dibujar el segundo texto en el lienzo se restaura el estado original, por lo que este segundo texto muestra los estilos predeterminados, no los declarados para el primero.

No importa cuántas transformaciones haya realizado, la configuración del lienzo volverá a ser exactamente la del estado previo a la aplicación del método restore().

10.1.10 globalCompositeOperation

Cuando hablamos de trazados dijimos que existe una propiedad que determina cómo se posiciona una forma y cómo se combina con figuras dibujadas previamente en el lienzo. La propiedad es globalCompositeOperation y su valor por defecto es source-over, lo que significa que la nueva forma será dibujada sobre las que ya existen en el lienzo. La propiedad ofrece 11 valores más:

source-in: solo es dibujada la parte de la nueva forma que se superpone a las formas previas. El resto de la figura, e incluso el resto de las figuras previas, se vuelven transparentes.

source-out: solo es dibujada la parte de la nueva forma que no se superpone a las figuras previas. El resto de la figura, e incluso el resto de las figuras previas, se vuelven transparentes.

source-atop: solo es dibujada la parte de la nueva forma que se superpone a las figuras previas, pero las figuras previas son preservadas y el resto de la nueva figura se vuelve transparente.

lighter: Ambas formas son dibujadas (la nueva y la vieja), pero el color de las partes que se superponen es igual a la suma de los valores de los colores de cada figura.

xor: Ambas figuras son dibujadas (la nueva y la vieja), pero las partes que se superponen se vuelven transparentes.

destination-over: Éste es el opuesto al valor por defecto. La nueva forma es dibujada detrás de las viejas que ya se encuentran en el lienzo.

destination-in: Las partes de las formas existentes en el lienzo que se superponen con la nueva figura son preservadas. El resto, incluyendo la nueva forma, se vuelven transparentes.

destination-out: Las partes de las formas existentes en el lienzo que no se superponen con la nueva forma son preservadas. El resto, incluyendo la nueva forma, se vuelven transparentes.

destination-atop: Las figuras existentes y la nueva son preservadas solo en la parte en la que se superponen.

darker: Ambas figuras son dibujadas, pero el color de las partes que se superponen es determinado substrayendo los valores de los colores de cada figura.

copy: solo la nueva figura es dibujada, mientras las ya existentes se vuelven transparentes.

```
function iniciar(){
  var elemento = document.getElementById('lienzo');

  car lienzo = elemento.getContext('2d');

  lienzo.fillStyle="#666666";
  lienzo.fillRect(100, 100, 200, 80);
  lienzo.globalCompositeOperation="source-atop";

  lienzo.fillStyle="#DDDDDD";
  lienzo.font="bold 60px verdana, sans-serif";
  lienzo.textAlign="center";
  lienzo.textBaseline="middle";
  lienzo.fillText("PRUEBA", 200, 100);
}
addEventListener("load", iniciar);
```

Código 10-21

Prueba de globalCompositeOperation.

Solo las representaciones visuales de cada valor posible de la propiedad globalCompositeOperation le ayudará a entender cómo funcionan, así que se preparó el ejemplo en el Código 10-21 por esta razón. Cuando se ejecuta este código, un rectángulo rojo se dibuja en el centro de la tela, pero como resultado del valor source-atop, solo la parte de la texto que solapa el rectángulo se dibuja en la pantalla.

Figura 10-13

Aplicación de globalCompositeOperation.

Hágalo usted mismo

Sustituya el valor source-atop con cualquiera de los otros valores de esta propiedad y compruebe el resultado en su navegador. Ponga a prueba su código en distintos navegadores.

Importante

En el momento en el que escribimos este manual, solo Mozilla Firefox produce los efectos adecuados para cada valor de esta propiedad al trabajar con texto. Para probar cada valor usando Google Chrome, reemplace el texto por una ruta o una figura.

10.3 Procesamiento de Imágenes

La API Canvas no sería lo que es sin la capacidad de procesamiento de imágenes pero, a pesar de la importancia de las imágenes, solo existe un método para este propósito.

10.3.1 drawImage()

El método drawImage() es el único que permite dibujar una imagen en el lienzo. Sin embargo, este método puede tomar una serie de valores que producen diferentes resultados. Echemos un vistazo a todas las posibilidades:

drawImage(imagen, x, y): Esta sintaxis dibuja una imagen en el lienzo en la posición declarada por x y y. El primer valor se refiere a la imagen que será dibujada y puede ser una referencia a un elemento <img>, una elemento <video> u otro elemento <canvas>.

drawImage(imágen, x, y, ancho, alto): Esta sintaxis permite escalar la imagen antes de dibujarla en el lienzo, cambiando su tamaño con los valores de los atributos ancho y alto.

drawImage(imagen, x1, y1, ancho1, alto1, x2, y2, ancho2, alto2): Ésta es la sintaxis más compleja. Hay dos valores para cada parámetro. El propósito es cortar partes de la imagen y luego dibujarlas en el lienzo con la posición y el tamaño indicados. Los valores x1 y y1 declaran la esquina superior izquierda de la parte de la imagen que será cortada. Los valores ancho1 y alto1 indican el tamaño de esta pieza. El resto de los valores (x2, y2, ancho2 y alto2) declaran el lugar donde la pieza será dibujada en el lienzo y su nuevo tamaño (que puede ser diferente al original).

En cada caso, el primer atributo puede ser una referencia a una imagen, un vídeo o un lienzo en el mismo documento. Pueden usarse referencias generadas por métodos, como getElementById(), o crearse un nuevo objeto imagen usando

métodos regulares de Javascript, como veremos en el siguiente ejemplo. No es posible usar una URL o cargar un archivo desde una fuente externa directamente con este método.

```
function iniciar(){
 var elemento = document.getElementById('lienzo');
 var lienzo = elemento.getContext('2d');


var imagen= document.createElement('img');
imagen.setAttribute('src', 'http://www.minkbooks.com/content/snow.
 jpg');
imagen.addEventListener("load", function(){
 lienzo.drawImage(imagen,20,20)
});
}
addEventListener("load", iniciar);
```

Código 10-22

Dibujar una imagen.

El Código 10-22 solo carga la imagen y la dibuja en el lienzo. Debido a que el lienzo solo puede dibujar imágenes que han sido previamente cargadas, necesitamos controlar esta situación con el evento load.

Agregamos un detector para este evento y declaramos el método drawImage() para responder al mismo. Cuando la imagen se ha cargado, se dibuja en la posición 20, 20.

Conceptos básicos

En el Código 7-22, dentro del método addEventListener() usamos una función anónima en lugar de una referencia a una función normal. En casos como éste, cuando la función es pequeña, esta técnica hace al código más simple y fácil de entender. Para aprender más sobre este tema, vaya a nuestro sitio web y visite los enlaces correspondientes a este capítulo.

Métodos como addEventListener() requieren una función como atributo. Para poder ejecutar el método, se tiene que incluir una función anónima. La función anónima es llamada cuando se activa el evento y a continuación se ejecutan los métodos indicados desde la función.

```
function iniciar(){
 var elemento = document.getElementById('lienzo');
 var lienzo = elemento.getContext('2d');


var imagen= document.createElement('img');
imagen.setAttribute('src', 'http://www.minkbooks.com/content/snow.
 jpg');
imagen.addEventListener("load", function(){
 lienzo.drawImage(imagen, 0, 0, elemento.width, elemento.height);
});
}
addEventListener("load", iniciar
```

Código 10-23

Ajuste de la imagen al tamaño del lienzo.

En el Código 10-23, se añaden dos valores al método drawImage() para cambiar el tamaño de la imagen. Las propiedades width y height devuelven las medidas del lienzo y la imagen se estira para cubrir todo el lienzo.

```
function iniciar(){
 var elemento = document.getElementById('lienzo');
 var lienzo = elemento.getContext('2d');

 var imagen = document.createElement('img');
 imagen.src="http://www.minkbooks.com/content/snow.jpg";

imagen.addEventListener("load", function(){
 lienzo.drawImage(imagen,135,30,50,50,0,0,200,200)
});
}
addEventListener("load", iniciar);
```

Código 10-24

Extraer, redimensionar y dibujar.

En el ejemplo del Código 10-24 se presenta una sintaxis más compleja del método drawImage(). Se proporcionan nueve valores para obtener una parte de la imagen original, cambiar su tamaño y luego dibujarla en el lienzo. Tomamos un cuadrado de la imagen original desde la posición 135, 50, con un tamaño de 50, 50 píxeles. Este bloque fue redimensionado a 200, 200 píxeles y finalmente dibujado en el lienzo en la posición 0, 0.

10.3.2 Datos de imagen

Cuando dijimos previamente que drawImage() era el único método disponible para dibujar imágenes en el lienzo, mentimos. Existen poderosos métodos para procesar imágenes que también pueden dibujarse en el lienzo. Sin embargo, estos métodos funcionan con datos, no imágenes y, ¿por qué íbamos a querer procesar los datos en lugar de imágenes?

Toda imagen puede ser descrita como una sucesión de números enteros que representan valores RGBA. Cuatro valores para cada píxel que representan los colores rojo, verde, azul y alfa (transparencia). Esta información constituye una matriz unidimensional que puede ser usada para generar la imagen. La API Canvas ofrece tres métodos para manipular datos y procesar imágenes de este modo:

getImageData(x, y, ancho, alto): Este método toma un rectángulo del lienzo del tamaño declarado por sus atributos y lo convierte en datos. Retorna un objeto con las propiedades width, height y data.

putImageData(datosImagen, x, y): Este método convierte los datos declarados por el atributo ImageData en una imagen y dibuja la imagen en el lienzo en la posición especificada por x e y. Hace lo opuesto a getImageData().

createImageData(ancho, alto): Este método crea datos para representar una imagen vacía. Todos los píxeles de la imagen serán de color negro transparente. Puede también recibir datos de imagen como atributo (en

lugar de los atributos ancho y alto) y utilizar las dimensiones de las imágenes cuyos datos han sido proporcionados.

La posición de cada valor en la matriz es calculada mediante la fórmula siguiente: (ancho × 4 × y) + (x × 4)+ n donde n es un número de índice para los valores del píxel, a partir de 0. El primer valor es el rojo y por tanto su fórmula es (ancho × 4 × y) + (x × 4) + 0. Para el verde es (ancho × 4 × y) + (x × 4) + 1; para azul (ancho × 4 × y) + (x × 4) + 2, y para el valor alpha es (ancho × 4 × y) + (x × 4) + 3. Veamos esto en práctica:

```
var lienzo, imagen;
function iniciar(){
 var elemento =  document.getElementById('lienzo');
 lienzo = elemento.getContext('2d');

 imagen = document.createElement('img');
 imagen.setAttribute('src', 'snow.jpg');
 imagen.addEventListener("load", modificarimagen);
}
function modificarimagen(){

 lienzo.drawImage(imagen,0,0);
 var info=lienzo.getImageData(0,0,175,262);
 var pos;
 for(var x=0; x<=175;x ++){
 for(var y=0; y<=262; y++){
  pos=(info.width*4*y)+(x*4);

  info.data[pos]=255-info.data[pos];
  info.data[pos+1]=255-info.data[pos+1];
  info.data[pos+2]=255-info.data[pos+2];
}
}
lienzo.putImageData(info,0,0);
}
addEventListener("load", iniciar);
```

Código 10-25

Generar un negativo de la imagen.

En el ejemplo del Código 10-25 creamos una nueva función para procesar la imagen después de que ésta se ha cargado. La función modimage() dibuja la imagen sobre el fondo en la posición 0, 0 usando el método drawImage(). De momento no hay nada inusual en esta parte del código, pero sigamos adelante.

Importante

Estos métodos presentan limitaciones para el acceso desde orígenes múltiples y por eso deberá subir los archivos usados en este ejemplo a su propio servidor, incluyendo la imagen llamada snow.jpg, que puede descargar desde www.minkbooks.com/content/snow.jpg).

La imagen utilizada en nuestro ejemplo tiene un tamaño de 350 píxeles de ancho por 262 píxeles de alto, por lo que usando el método getImageData() con los valores 0, 0 para la esquina superior izquierda y 175, 262 para las dimensiones horizontal y vertical, extraemos solo la mitad izquierda de la imagen original. Estos datos son grabados dentro de la variable info y luego se procesan para obtener el efecto deseado. En este ejemplo vamos a crear un negativo de este trozo de la imagen.

Debido a que cada color es declarado por un valor entre 0 y 255, el valor negativo es obtenido restando el valor real a 255 con la fórmula color = 255 - color. Para hacer esto con cada píxel de la imagen, debemos crear dos bucles for, uno para las columnas y otro para las filas. Observe que el bucle for para los valores x va desde 0 a 174 (el ancho de la parte de la imagen que extrajimos del lienzo) y el for para los valores y va desde 0 a 261 (el número total de píxeles horizontales de la imagen que se está procesando).

Luego de que cada píxel es procesado, la variable info con los datos de la imagen es enviada al lienzo como una imagen usando el método putImageData(). La imagen es ubicada en la misma posición que la original, reemplazando la mitad izquierda de la imagen original por el negativo que acaba de crear.

Figura 10-14

Procesar datos de imagen.

10.3.3 cross-Origin

Una aplicación de orígenes mixtos es aquella que se asigna en un dominio pero accede a los recursos de otro dominio. Por ejemplo, cuando se probaron los ejemplos para los elementos <video> o <audio> en los capítulos anteriores, se utilizó una aplicación de dos dominios: el archivo con el documento HTML estaba en su ordenador (o servidor), mientras que los archivos de vídeo y audio se almacenaban en los servidores de MinkBooks.

Por motivos de seguridad, algunas API restringen el acceso a contenidos de orígenes múltiples. En el caso de la API Canvas: ninguna información puede ser recuperada del elemento <canvas> después de que es dibujada una imagen de otro dominio.

Estas restricciones pueden ser evitadas mediante una nueva tecnología llamada CORS (algo así como “Intercambio de recursos de origen mixto”). CORS define un protocolo para que servidores compartan sus recursos y permitan peticiones de otros orígenes. De acuerdo con esta especificación de la tecnología, el acceso de un origen a otro debe ser autorizado por el servidor. La autorización se realiza mediante la declaración de los orígenes (dominios) autorizados a acceder a los recursos. Esto se hace en el encabezado enviado por el servidor que aloja el archivo que procesa la solicitud.

Por ejemplo, si la aplicación está ubicada en www.dominio1.com y está accediendo a recursos de www.dominio2.com, este segundo servidor debe estar configurado para declarar el origen www.dominio1.com como un origen válido para la solicitud.

CORS ofrece varios encabezados para ser incluidos como parte de la cabecera HTTP enviada por el servidor, pero el único necesario es Access-Control-Allow-Origin. Este encabezado indica qué orígenes (dominios) son capaces de acceder a los recursos del servidor. Puede ser declarado el carácter * para permitir peticiones de cualquier origen.

Importante

Su servidor debe estar configurado para enviar cabeceras HTTP CORS con cada solicitud. Una manera fácil de hacer esto es añadir una nueva línea al archivo .htaccess. La mayoría de los servidores ofrecen este archivo de configuración en la carpeta raíz de cada sitio web. La sintaxis correspondiente es Header set CORS-Header value (por ejemplo Header set Access-Control-Allow-Origin *). Para obtener más información sobre cómo añadir cabeceras HTTP al servidor, por favor visite nuestro sitio web y siga los enlaces de este capítulo.

Añadir cabeceras HTTP a la configuración del servidor es solo uno de los pasos que tenemos que seguir para convertir una secuencia de comandos en una aplicación de origen múltiple. También hay que declarar el recurso como un recurso de origen usando el atributo crossOrigin.

```
var lienzo, imagen;
function iniciar(){
 var elemento =  document.getElementById('lienzo');
 lienzo = elemento.getContext('2d');

 imagen = document.createElement('img');
 imagen.setAttribute('crossOrigin', 'anonymous');
 imagen.setAttribute('src', 'http://www.minkbooks.com/content/snow.
 jpg');
   imagen.addEventListener("load", modificarimagen);
}
function modificarimagen(){

 lienzo.drawImage(imagen,0,0);
 var info=lienzo.getImageData(0,0,175,262);
 var pos;
 for(var x=0; x<=175;x ++){
 for(var y=0; y<=262; y++){
  pos=(info.width*4*y)+(x*4);
  info.data[pos]=255-info.data[pos];
  info.data[pos+1]=255-info.data[pos+1];
  info.data[pos+2]=255-info.data[pos+2];
}
}
lienzo.putImageData(info,0,0);
}
addEventListener("load", iniciar);
```

Código 10-26

Acceso desde orígenes múltiples.

El atributo crossOrigin puede tomar dos valores: anonymous o use-credentials. El primer valor hace caso omiso de las credenciales, mientras que el segundo valor requiere credenciales para ser enviado en la petición. Las credenciales son compartidas automáticamente por el cliente y el servidor utilizando cookies. Para poder utilizar las credenciales tenemos que incluir una segunda cabecera en el servidor llamada Access-Control-Allow-Credentials con el valor true.

Importante

El atributo crossOrigin debe ser declarado antes del atributo src para establecer la fuente como de origen transversal antes que el archivo se haya descargado por el navegador. Por supuesto, el atributo también puede ser declarado en la etiqueta de apertura de los elementos <img>, <video> y <audio>, como cualquier otro atributo HTML.

En el Código 10-26, solo se hizo una modificación a partir del ejemplo anterior: se agregó el atributo crossOrigin al elemento img para declarar la imagen como un recurso de origen transversal. Ahora se puede ejecutar la secuencia de comandos en el servidor y utilizar la imagen del servidor MinkBooks sin violar la política de mismo origen (ya se han añadido las cabeceras CORS a nuestro servidor).

10.3.4 Extracción de los datos

El método getImageData() previamente estudiado retorna un objeto que puede ser procesado a través de sus propiedades (width, height y data) o puede ser usado íntegro por el método putImageData(). El propósito de estos métodos es proporcionar acceso a los datos contenidos en el lienzo y devolver los datos al mismo lienzo o a otro después de ser procesados. A veces, es posible que necesitemos la información para otros fines, como por ejemplo su uso para la fuente de un elemento HTML como <img>, enviar el contenido del lienzo a un servidor o almacenar los datos en un archivo. Para realizar estas tareas, la API Canvas incluye los siguientes métodos:

toDataURL(tipo): Este método retorna datos en el formato data:url conteniendo una representación del contenido del lienzo en el formato especificado por el atributo tipo y a una resolución de 96 ppp. Si no hay ningún tipo declarado, los datos se devuelven como PNG. Los valores posibles para el atributo son image/jpeg y image/png.

toDataURL(): Este método es similar al anterior, pero los objetos devueltos tienen la resolución original del liezo.

toBlob(funcion, tipo): Este método devuelve un objeto con un blob (datos “en crudo”) que contienen una representación del contenido del lienzo en el formato especificado por el atributo tipo y una resolución de 96 ppp. El primer atributo es la función encargada de procesar el objeto. Si no se declara ningún tipo, el blop se devuelve como PNG. Los valores posibles para el atributo tipo son image/jpeg e image/png.

toBlobHD(tipo): Es similar al método anterior, pero el blob devuelto tiene la resolución original del lienzo.

Debido a que estos métodos están destinados a ser utilizados con otras API, vamos a ver algunos ejemplos más adelante en este libro.

Conceptos básicos

data:url y blobs son dos formatos diferentes para la distribución de datos. El formato data:url es oficialmente llamado Esquema de datos URI. Este esquema es una manera de incluir datos en un documento para imitar a recursos externos, tales como una imagen para el elemento <img> (véase el Capítulo 16, Código 16-4 para un ejemplo). Este formato es una cadena de texto que representa datos para crear el recurso. Blobs, por otro lado, son los bloques que contienen datos brutos. Vamos a estudiar los blobs en el Capítulo 16.

10.3.5 Patrones

Los patrones son simples adiciones que pueden mejorar nuestros trazados. Permiten agregar textura a las formas creadas utilizando una imagen. El procedimiento es similar a la creación de degradados; los patrones son creados por el método createPattern() y luego aplicados al trazado como si fuesen un color con la propiedad fillStyle.

createPattern(imagen, tipo): El atributo imagen es una referencia a la imagen que vamos a usar como patrón, y el tipo configura el patrón por medio de cuatro valores: repeat, repeat-x, repeat-y y no-repeat.

```
var canvas, img;
function iniciar(){
 var elemento = document.getElementById('lienzo');
 lienzo = elemento.getContext('2d');

 imagen= document.createElement('img');
 imagen.setAttribute('src', 'http://www.minkbooks.com/content/
 bricks.jpg');
 imagen.addEventListener("load", modificarimagen);
}

function modificarimagen(){
var patron=lienzo.createPattern(imagen,'repeat');
 lienzo.fillStyle=patron;
 lienzo.fillRect(0,0,500,300);
}
addEventListener("load", iniciar);
```

Código 10-27

Adición de un modelo para nuestros patrones.

Hágalo usted mismo

Experimente con los diferentes valores disponibles para createPattern() y asigne patrones a otras formas.

10.4 Animaciones sobre lienzo

Las animaciones son creadas por código Javascript convencional. No existen métodos para ayudarnos a animar figuras en el lienzo, y tampoco existe un procedimiento predeterminado para hacerlo. Básicamente, debemos borrar el área del lienzo que queremos animar, dibujar las figuras y repetir el proceso una y otra vez. Una vez que las figuras son dibujadas no se pueden mover. solo borrando el área y dibujando las figuras nuevamente podemos construir una animación. Por esta razón, en juegos o aplicaciones que requieren grandes cantidades de objetos a ser animados, es mejor usar imágenes en lugar de figuras construidas con trazados complejos (por ejemplo, los juegos que normalmente utilizan imágenes PNG).

10.4.1 Animaciones elementales

Existen múltiples técnicas para lograr animaciones en el mundo de la programación. Algunas son simples y otras tan complejas como las aplicaciones para las que fueron creadas. Vamos a ver un ejemplo simple utilizando el método clear-Rect() para limpiar el lienzo y dibujar una nueva forma.

```
var lienzo;
function iniciar(){
 var elemento = document.getElementById('lienzo');
 lienzo = elemento.getContext('2d');
 addEventListener('mousemove', animacion);
}

 function animacion(e){
  lienzo.clearRect(0,0,300,500);

  var xraton = e.clientX;
  var yraton = e.clientY;
  var xcentro = 220;
  var ycentro = 150;
  var angulo = Math.atan2(xraton-xcentro,yraton-ycentro);
  var x = xcentro+Math.round(Math.sin(angulo)*10);
  var y = ycentro+Math.round(Math.cos(angulo)*10);

  lienzo.beginPath();
  lienzo.arc(xcentro,ycentro,20,0,Math.PI*2, false);
  lienzo.moveTo(xcentro+70,150);
  lienzo.arc(xcentro+50,150,20,0,Math.PI*2, false);
  lienzo.stroke();

  lienzo.beginPath();
  lienzo.moveTo(x + 10,y);
  lienzo.arc(x,y,10,0,Math.PI*2, false);
  lienzo.moveTo(x+60,y);
  lienzo.arc(x+50,y,10,0,Math.PI*2, false);
  lienzo.fill();
}
addEventListener("load", iniciar);
```

Código 10-28

Creando nuestra primera animación.

El Código 10-28 mostrará dos ojos en pantalla que miran al puntero del ratón todo el tiempo. Para mover los ojos debemos actualizar su posición cada vez que se mueve el ratón. Por este motivo agregamos un detector para el evento mousemove en la función iniciar(). Cuando el evento es disparado, la función animacion() es llamada.

Importante

En este ejemplo, la distancia se calcula sin tener en cuenta la posición del lienzo en la pantalla porque el elemento <canvas> fue creado en la esquina superior izquierda de la página (la esquina superior izquierda del lienzo y la esquina superior izquierda del documento son lo mismo). En el siguiente ejemplo, una animación profesional, el lienzo está centrado en la ventana y la fórmula de la distancia se amplía para considerar esta nueva ubicación.

La función animacion() limpia el lienzo con la instrucción clearRect(0, 0, 300, 500) y luego, inicia las variables. La posición del puntero del ratón es capturada usando las viejas propiedades clientX y clientY y la posición del primer ojo es grabada en las variables xcentro e ycentro. Luego de que los valores de estas variables son declarados, es tiempo de comenzar con las matemáticas. Usando los valores de la posición del ratón y el centro del ojo izquierdo, calculamos el ángulo de la línea invisible que va desde un punto al otro usando el método

Javascript predefinido atan2, que pertenece al objeto Math que se explicó antes. Este ángulo es usado en el siguiente paso para calcular el punto exacto del centro del iris del ojo izquierdo con la fórmula xcentro + Math.round(Math.sin(angulo) × 10). El número 10 en la fórmula representa la distancia desde el centro del ojo al centro del iris (porque el iris no está en el centro del ojo, está siempre sobre el borde).

Con todos estos valores podemos finalmente comenzar a dibujar nuestros ojos en el lienzo. El primer trazado es para los dos círculos que representan los ojos. El primer método arc() para el primer ojo es posicionado en los valores xcentro y ycentro, y el círculo para el segundo ojo es generado 50 píxeles hacia la derecha usando la instrucción arc(xcentro+50, 150, 20, 0, Math.PI*2, false).

Hágalo usted mismo

Copie el Código 10-28 en el archivo Javascript llamado canvas.js y abra el archivo HTML con el Código 10-1 en su navegador.

La parte animada del gráfico es creada a continuación con el segundo trazado. Este trazado usa las variables x e y con la posición calculada previamente a partir del ángulo. Ambos iris son dibujados como un círculo negro sólido usando el método fill(). El proceso será repetido y los valores recalculados cada vez que el evento mousemove es disparado (es decir, cada vez que el usuario mueve el ratón).

Conceptos básicos

clientX y clientY son propiedades tradicionales de eventos que devuelven la posición del puntero del ratón en relación con la pantalla en el momento en el que el evento tiene lugar.

10.4.2 Animaciones profesionales

La animación en bucle del ejemplo anterior ha sido generada generado por el evento mousemove. En una animación profesional, los bucles se ejecutan independientemente de la respuesta del usuario. En estas animaciones complejas los bucles son controlados por el código. Javascript ofrece métodos tradicionales para ejecutar un bucle, tales como setInterval() y setTimeout(). Estos métodos ejecutan una función después de un período específico de tiempo. Esto produce animaciones decentes, pero que no están sincronizadas con el navegador, causando problemas técnicos que no son tolerados en una aplicación profesional. Para resolver este problema, HTML5 introduce una pequeña API con solo dos métodos: uno para generar un nuevo ciclo para el bucle y otro para cancelarlo.

requestAnimationFrame (función): Este método envía al navegador una señal de que la función entre paréntesis debe ser ejecutada. El explorador llama a la función cuando está listo para actualizar la ventana, sincronizando así la animación con la ventana del navegador y la pantalla del ordenador. Podemos asignar este método a una variable y luego cancelar el proceso utilizando cancelAnimationFrame() con el nombre de la variable entre paréntesis.

El método requestAnimationFrame() funciona como setTimeout(); es decir, tenemos que llamarlo para cada ciclo del bucle. La implementación es sencilla, pero el Código para una animación profesional requiere cierta organización que solo se puede lograr con programación avanzada. Para nuestro ejemplo vamos a usar un objeto global y distribuir las tareas entre varios métodos.

Para comenzar, vamos a crear un documento HTML para proporcionar el elemento <canvas> y algunos estilos.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>vídeo juego</title>
  <style>
    body{
      text-align: center;
    }
    #Cajadelienzo{
      margin: 100px auto;
    }
    #lienzo{
      border: 1px solid #999999;
    }
  </style>
  <script src="videojuego.js"></script>
</head>
<body>
  <section id="Cajadelienzo">
    <canvas id="lienzo" width="600" height="400"></canvas>
  </section>
</body>
</html>
```

Código 10-29

Documento HTML para nuestro videojuego.

Los estilos CSS en el documento de Código 10-29 tienen el propósito de centro de la tela en la pantalla y proporcionar un borde para identificar sus límites. El documento también carga un archivo de Javascript con el siguiente código.

```
var mijuego = {
  lienzo: {
    ctx: '',
    offsetx: 0,
    offsety: 0
  },
  nave:{
    x: 300,
    y: 200,
    movex: 0,
    movey: 0,
    speed: 1
  },
  iniciar: function(){
    var elem = document.getElementById('lienzo');
    mijuego.lienzo.ctx = elem.getContext('2d');
    mijuego.lienzo.offsetx = elem.offsetLeft;
    mijuego.lienzo.offsety = elem.offsetTop;
    document.addEventListener('click', function(e){ mijuego.
                        control(e);});
    mijuego.bucle();
  },
  bucle: function(){
    if(mijuego.nave.speed){
      mijuego.procesar();
      mijuego.detectar();
      mijuego.dibujar();
      webkitRequestAnimationFrame(function(){ mijuego.bucle() });
    }else{
      mijuego.lienzo.ctx.font = "bold 36px verdana, sans-serif";
      mijuego.lienzo.ctx.fillText('Final del juego', 170, 210);
    }
  },
  control: function(e){
    var distancex = e.clientX - (mijuego.lienzo.offsetx + mijuego.
                        nave.x);
    var distancey = e.clientY - (mijuego.lienzo.offsety + mijuego.
                        nave.y);
    var ang = Math.atan2(distancex, distancey);


    mijuego.nave.movex = Math.sin(ang);
    mijuego.nave.movey = Math.cos(ang);
    mijuego.nave.speed += 1;
  },
  dibujar: function(){
    mijuego.lienzo.ctx.clearRect(0, 0, 600, 400);
    mijuego.lienzo.ctx.beginPath();
    mijuego.lienzo.ctx.arc(mijuego.nave.x, mijuego.nave.y, 20, 0,
                        Math.PI/180*360, false);
    mijuego.lienzo.ctx.fill();
  },
  procesar: function(){
    mijuego.nave.x += mijuego.nave.movex * mijuego.ship.speed;

    mijuego.nave.y += mijuego.nave.movey * mijuego.nave.speed;
  },
  detectar: function(){
    if(mijuego.nave.x < 0 || mijuego.nave.x > 600 || mijuego.nave.y
                        < 0 || mijuego.nave.y > 400){
      mijuego.nave.speed = 0;
    }
  }
};
addEventListener('load', function(){ mijuego.iniciar(); });
```

Código 10-30

La creación de un videojuego en 2D.

Importante

Ésta es una API experimental. Los métodos han sido prefijados por los navegadores mientras se implementa la especificación final. De momento, Google Chrome y Mozilla Firefox son los navegadores que tienen una implementación funcional del método requestAnimationFrame(). Google Chrome lo ha implementado como webkitRequestAnimationFrame() y Mozilla Firefox como mozRequestAnimationFrame(). El último ejemplo aplica la versión experimental de Google Chrome. Para ejecutar este script en un navegador diferente, tiene que declarar el método usando el prefijo correspondiente.

Una aplicación profesional debe evitar las variables y funciones globales y concentrar el código dentro de un objeto global único. En el ejemplo del Código 10-30, llamamos a este objeto mijuego. Todas las propiedades y métodos necesarios para crear el pequeño videojuego se declaran dentro de este objeto.

El juego se trata de una nave espacial de color negro que se mueve en la dirección indicada por el clic del ratón. El objetivo es cambiar la dirección de la nave para evitar las paredes. Cada vez que se cambia la dirección, la velocidad de la nave aumenta, lo que hace más y más difícil de mantenerla en el interior del lienzo.

La organización necesaria para crear este tipo de aplicaciones tiene algunos elementos esenciales. Tenemos que declarar valores básicos, iniciarlos, controlar el bucle de animación y distribuir tareas. En el ejemplo del Código 10-30 esta organización se logra con la declaración de propiedades esenciales y la creación de seis métodos: iniciar(), bucle(), control(), dibujar(), procesar() y detectar().

Empezamos por declarar las propiedades lienzo y nave. Estas propiedades contienen objetos con información esencial para el juego. El objeto lienzo tiene tres propiedades: ctx para el contexto de la lona y, por otra parte, OffsetX y OffsetY para la posición del elemento <canvas> en relación a la página. El objeto nave, por otro lado, tiene cinco propiedades: x e y para las coordenadas de la nave, movex y movey para la dirección, y speed para la velocidad de la nave.

Esas propiedades son importantes y necesarias en casi todas las secciones del código, pero algunos de sus valores tienen que ser todavía inicializados. Esto se hace mediante el método llamado iniciar(), que es llamado por el evento load cuando el documento se ha cargado y es responsable de asignar todos los valores esenciales necesarios para iniciar la aplicación. En primer lugar se obtiene el contexto del lienzo y la posición del elemento en la ventana utilizando las propiedades offsetLeft y offsetTop. Entonces se añade un detector al evento clic para todo el documento con el propósito de obtener una respuesta del usuario. Al final se llama al método bucle() para iniciar el bucle de animación.

El método loop() es el segundo método más importante en la organización básica de una aplicación profesional. Este método se llamará una y otra vez mientras se ejecuta la aplicación, creando un bucle que pasa a través de cada parte del proceso. En nuestro ejemplo, este proceso se divide en tres métodos: procesar(), detectar() y dibujar(). El método procesar() calcula la nueva posición de la nave de acuerdo con la dirección y la velocidad, el método detectar() compara las coordenadas de la nave con los límites del lienzo para determinar si la nave se ha estrellado contra las paredes, y el método dibujar() dibuja la nave en el lienzo. El bucle ejecuta estos métodos uno a uno y después se llama a sí mismo con el

método requestAnimationFrame(), comenzando de nuevo.

Lo único que queda por hacer para terminar la estructura básica de nuestra aplicación es comprobar la retroalimentación del usuario. En nuestro juego, esto se hace mediante el método control(). Cuando el usuario hace clic en cualquier parte del documento, se llama a este método para calcular la dirección de la nave. La fórmula es similar a la del ejemplo anterior: primero, se calcula la distancia desde la nave hasta el lugar donde se produce el evento, a continuación, se obtiene el ángulo de la línea invisible entre esos dos puntos, y, finalmente, las coordenadas obtenidas por los métodos sin() y cos() se almacenan en las propiedades movex y movey. La última instrucción del método control() es simplemente un aumento de la velocidad de la nave para hacer que nuestro juego sea un poco más interesante.

El juego se inicia tan pronto como el documento se ha cargado y termina cuando la nave se estrella contra una pared. Para hacer que esta pequeña aplicación se parecezca más a un videojuego, se añade una declaración if al bucle que comprueba el estado de la nave. El estado está determinado por el valor de la velocidad. Cuando el procedimiento detect() determina que las coordenadas de la nave están fuera de los límites del lienzo, la velocidad se reduce a 0. Este valor devuelve el resultado false para el estado del bucle y entonces es mostrado en la pantalla el mensaje Fin del juego.

Importante

Como hemos explicado antes, métodos como requestAnimationFrame() y addEventListener() requieren que sea declarada una función como atributo. En nuestro ejemplo anterior tuvimos que utilizar una función anónima para llamar a los métodos del juego. Primero se llama a la función anónima y luego son llamados los métodos adecuados desde dentro de la función.

10.5 Procesar vídeo en el lienzo

Al igual que para las animaciones, no hay ningún método especial para mostrar los vídeos en el elemento <canvas>. La única manera de hacerlo es tomando cada cuadro del vídeo desde el elemento <video> y dibujarlo como una imagen en el lienzo usando el método drawImage(). Así que, básicamente, el procesamiento de vídeo en el lienzo es hecho con una combinación de las técnicas ya estudiadas.

10.5.1 Mostrar vídeo en el lienzo

Vamos a ver cómo funciona el proceso con un ejemplo sencillo. El siguiente documento incluye un breve código Javascript para convertir el lienzo en un espejo.

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Video en lienzo</title>
<style>
  section{
  float: left;
}
</style>
<script>
  var lienzo, video;
  function iniciar(){
  var elem = document.getElementById('lienzo');
  lienzo = elem.getContext('2d');
  video = document.getElementById('media');
  lienzo.translate(483, 0);
  lienzo.scale(-1, 1);
  setInterval(procesarCuadros, 33);
}
function procesarCuadros(){
  lienzo.drawImage(video, 0, 0);
}
addEventListener("load", iniciar);
</script>
</head>
```

Código 10-31

Mostrar vídeo sobre el lienzo.

Como ya se explicó, el método drawImage() puede tomar tres tipos de fuentes: una imagen, un vídeo u otro lienzo. Por lo tanto, para mostrar el vídeo en el lienzo se proporciona una referencia al elemento <video> como primer atributo del método. Sin embargo, hay que tener algo en cuenta: los vídeos se componen de varios cuadros y el método drawImage() solo es capaz de tomar un cuadro a la vez. Por consiguiente, para mostrar el vídeo completo en el lienzo hay que repetir el proceso para cada fotograma. En el Código 10-31, se utiliza un método setInterval() para llamar a la función procesarCuadros() cada 33 milisegundos. Esta función ejecuta el método drawImage() con el vídeo como fuente. El tiempo establecido por setInterval() es aproximadamente el tiempo que se toma cada cuadro en la pantalla.

Una vez que la imagen del cuadro está en el lienzo, está unido a las propiedades del lienzo y se puede procesar como contenido del lienzo. En nuestro ejemplo, la imagen se invierte para crear un efecto de espejo. El efecto es producido por la aplicación del método scale() al contexto del lienzo y todo lo dibujado en el lienzo se invierte.

10.5.2 Aplicación de la vida real

Hay un millón de cosas que hacer con el lienzo, pero siempre es fascinante ver lo lejos que podemos llegar combinando métodos sencillos de diferentes API. El siguiente ejemplo explica cómo tomar una fotografía con la cámara web y mostrarla en la pantalla utilizando <canvas>.

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>Aplicación Instantáneas</title>
<style>
  section{
  float: left;
  width: 320px;
  height: 240px;
  border: 1px solid #000000;
  }
</style>
<script>
  var video, lienzo;
  function iniciar(){
   navigator.webkitGetUserMedia({video: true}, exito, mostrarError);
  }
  function exito(stream){
   var elem = document.getElementById('canvas');
   lienzo = elem.getContext('2d');
   var reproductor = document.getElementById("reproductor");
   reproductor.addEventListener('click', instantanea);
   video = document.getElementById("media");
   video.setAttribute('src', URL.createObjectURL(stream));
   video.play();
  }
  function mostrarError(e){
   console.log(e);
  }
  function instantanea(){
   lienzo.drawImage(video, 0, 0, 320, 240);
  }
addEventListener('load', iniciar);
</script>
</head>
<body>
<section id="reproductor">
<video id="media" width="320" height="240"></video>
</section>
<section>
<canvas id="canvas" width="320" height="240"></canvas>
</section>
</body>
</html>
```

Código 10-32

Programación de una aplicación de instantáneas.

El código anterior ofrece algunos estilos CSS, el código Javascript y unos pocos elementos HTML, incluyendo los necesarios elementos <video> y <canvas>. El elemento <canvas> se declaró como de costumbre, pero no especificó la fuente del elemento <video> porque vamos a utilizarlo para mostrar una imagen de la webcam. Para organizar los dos elementos en la pantalla, los elementos <section> fueron declarados para generar dos cajas de lado a lado.

Importante

En este ejemplo se utiliza el método webkitGetUserMedia(), específico para Google Chrome. Éste es uno de los navegadores con una implementación de la API Stream (Capítulo 9), pero el método es todavía experimental, de manera que tiene que usar el prefijo. Para probar este ejemplo en un navegador diferente tendrá que utilizar el prefijo correspondiente.

El código es tan simple como potente. La función iniciar() llama al método getUserMedia() en cuanto el documento se ha cargado para tener acceso a la cámara web. En caso de éxito, se activa la función exito(). La mayor parte del trabajo es realizado por esta función. El contexto del lienzo se crea, se añade un detector para el evento click en la caja correspondiente al vídeo, y el vídeo de la cámara web se asigna al elemento <video>. Al final, el vídeo se reproduce utilizando el método play().

De momento tenemos el vídeo de la webcam en la caja de la izquierda. Para tomar una instantánea de ese vídeo en el cuadro de la derecha, se crea la función snapshot(). Esta función responde al evento click. Cuando el usuario hace clic en el vídeo, la función se activa. Aquí se llama al método de canvas drawImage() con una referencia al vídeo y a los valores correspondientes al tamaño del elemento <canvas>. El método toma el fotograma de vídeo actual y lo dibuja en el lienzo, tomando una instantánea.

Hágalo usted mismo

Cree un archivo HTML con el documento del Código 10-32. Para probar el ejemplo, debe cargar el archivo en el servidor.

11. WebGL y Three.js

11.1 Lienzo en 3D

WebGL es una API de muy bajo nivel que trabaja con el elemento <canvas> para crear gráficos en 3D para la Web. Utiliza la GPU (unidad de procesamiento gráfico) de la tarjeta de vídeo para realizar operaciones y libera a la CPU (unidad central de procesamiento) de trabajo. Se basa en OpenGL, una conocida biblioteca desarrollada por Silicon Graphics Inc. que ha sido utilizada para la creación de exitosos vídeojuegos y aplicaciones 3D desde 1992. Estas características dan confianza y eficiencia a WebGL, por ese motivo se ha convertido en la API 3D estándar para la Web, aunque no ha sido oficialmente declarada como parte de la especificación HTML5.

Importante

Debido a las características de WebGL y a lo extenso que es el tema, no vamos a estudiar y aplicar WebGL en sí misma. En este capítulo sólo estudiaremos cómo generar y trabajar con gráficos en 3D usando una simple biblioteca externa llamada Three.js.

WebGL ha sido implementada en casi todos los navegadores compatibles con HTML5, pero su complejidad ha obligado a los desarrolladores trabajar con librerías Javascript construidas sobre ésta, en lugar de la API en sí misma. Esta complejidad va más allá del hecho de que WebGL no incorpora métodos nativos para realizar las operaciones básicas en 3D sino que requiere la inserción de códigos externos programados en un lenguaje llamado GLSL (OpenGL Shading Language) para proporcionar características esenciales para la producción de la imagen en la pantalla. Por estas razones, el desarrollo de una aplicación con

WebGL siempre requiere el uso de bibliotecas externas, desde glMatrix para las operaciones con matrices y vectores (github.com/toji/gl: matrix), hasta bibliotecas de propósito más general, como Three.js (www.threejs.org), GLGE (www.glge.org), SceneJS (www.scenejs.org), entre otras.

11.2 Three.js

Three.js es probablemente la biblioteca más popular y potente disponible para la generación de gráficos 3D mediante WebGL. Esta biblioteca funciona sobre WebGL, lo que simplifica la mayoría de las tareas y proporciona las herramientas necesarias para el control de cámaras, luces, objetos, texturas y mucho más.

Importante

Este capítulo no pretende ser un manual para Three. js. Para una referencia más completa, consulte la documentación sobre la biblioteca disponible en www.threejs.org.

Al igual que otras bibliotecas externas, es fácil de instalar: solo hay que descargar el paquete de www.threejs.org e incluir el archivo three.min.js en la carpeta del documento HTML.

La biblioteca incluye varios constructores para generar los objetos básicos requeridos para trabajar con gráficos en 3D. Estos objetos proporcionan los métodos y propiedades necesarios para establecer la escena, la cámara y las mallas que representan a los objetos físicos y también para trabajar en estos elementos con el fin de construir y animar el mundo tridimensional en la Web.

11.2.1 Renderer

El renderer es el procesador, la superficie donde se dibujan los gráficos. Three. js utiliza un elemento <canvas> con un contexto webGL para representar escenas 3D en el navegador (que crea el contexto con el parámetro WebGL). Para configurar este procesador, la biblioteca ofrece el constructor WebGLRenderer():

WebGLRenderer(parámetros): Este constructor devuelve un objeto con propiedades y métodos para configurar la superficie de dibujo y representar los gráficos en la pantalla. El atributo parámetros debe ser especificado como un objeto con propiedades específicas. Las propiedades disponibles actualmente son canvas (lienzo del elemento), precision (valores highp, mediump, lowp), alpha (valor booleano), premultipliedAlpha (valor booleano), antialias (valor booleano), stencil (valor booleano), preserveDrawingBuffer (valor booleano), clearColor (valor entero), y clearAlpha (valor decimal). En los ejemplos de este capítulo vamos a configurar el procesador utilizando algunas de estas propiedades.

El objeto WebGLRenderer proporciona varios métodos y propiedades para establecer y obtener la información del renderer. Los más comúnmente utilizados son:

setSize(ancho, alto): Este método cambia el tamaño del lienzo para los valores width y height.

setViewport(x, y, ancho, alto): Este método determina el área del lienzo que se utiliza para representar la escena. El tamaño del área utilizada por

WebGL no necesariamente tiene que ser el mismo que el de la superficie de dibujo. Los atributos x e y indican las coordenadas de partida para la vista mientras ancho y alto indican su tamaño.

setClearColorHex(color, alfa): Este método establece un color para la superficie en valores hexadecimales. El atributo alfa declara la opacidad (de 0,0 a 1,0).

render(escena, camara, objetivo, limpiar): Este método representa la escena usando una cámara. Los atributos escena y camara son objetos que representan la escena y la cámara. El atributo objetivo declara dónde se llevará a cabo la representación (no es necesario si queremos usar el lienzo establecido por el constructor del procesador), y el atributo booleano limpiar determina si el lienzo tiene que ser limpiado antes de la representación o no.

Importante

La biblioteca incluye también el constructor CanvasRenderer(), que devuelve un objeto para trabajar con un contexto de lienzo 2D cuando el navegador no es compatible con WebGL. El procesador del lienzo no funciona con la GPU y no se recomienda para aplicaciones de alto nivel.

11.2.2 scene

Scene es un objeto global que contiene el resto de los objetos que representan cada elemento del mundo 3D, tales como la cámara, luces, mallas, etc. Three.js proporciona un sencillo constructor para generar una escena:

scene(): Este constructor devuelve un objeto que representa la escena. Un objeto Scene ofrece los métodos add() y remove() para añadir y eliminar elementos de la escena. La escena establece un espacio tridimensional que ayuda a ubicar cada elemento en el mundo virtual. Las coordenadas de este espacio se identifican con x, y y z. Cada elemento tiene sus propios valores de coordenadas y posición en el mundo 3D, tal como se representa en la siguiente figura.

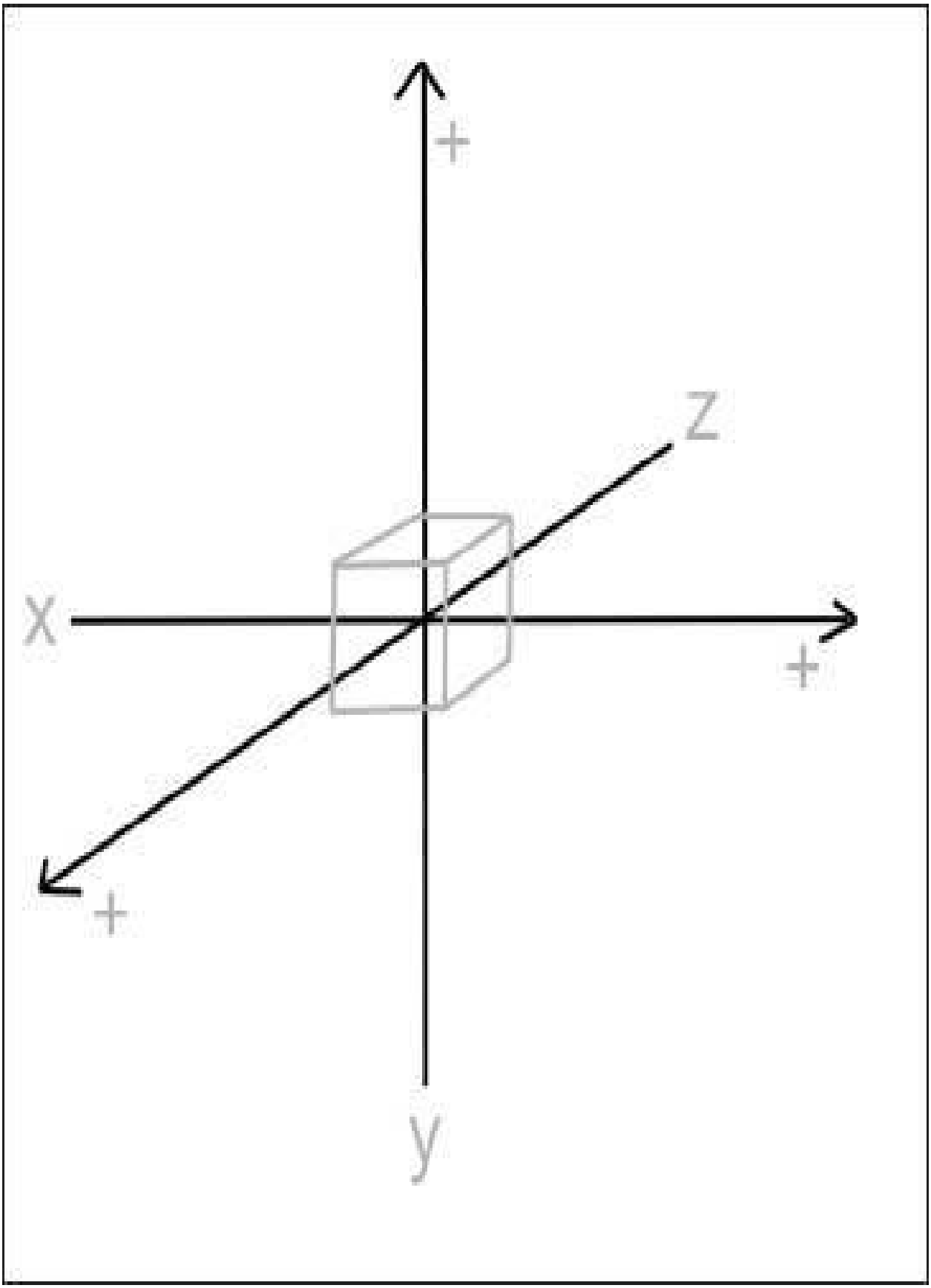

+
Z
X
+
+
y

Figura 11-1

Un cubo 3D en un sistema de coordenadas.

Si se aumenta el valor de la coordenada x para un elemento, este elemento se desplazará a una nueva posición en el eje x, manteniendo la misma posición en el resto de los ejes, como se muestra en la Figura 11-2.

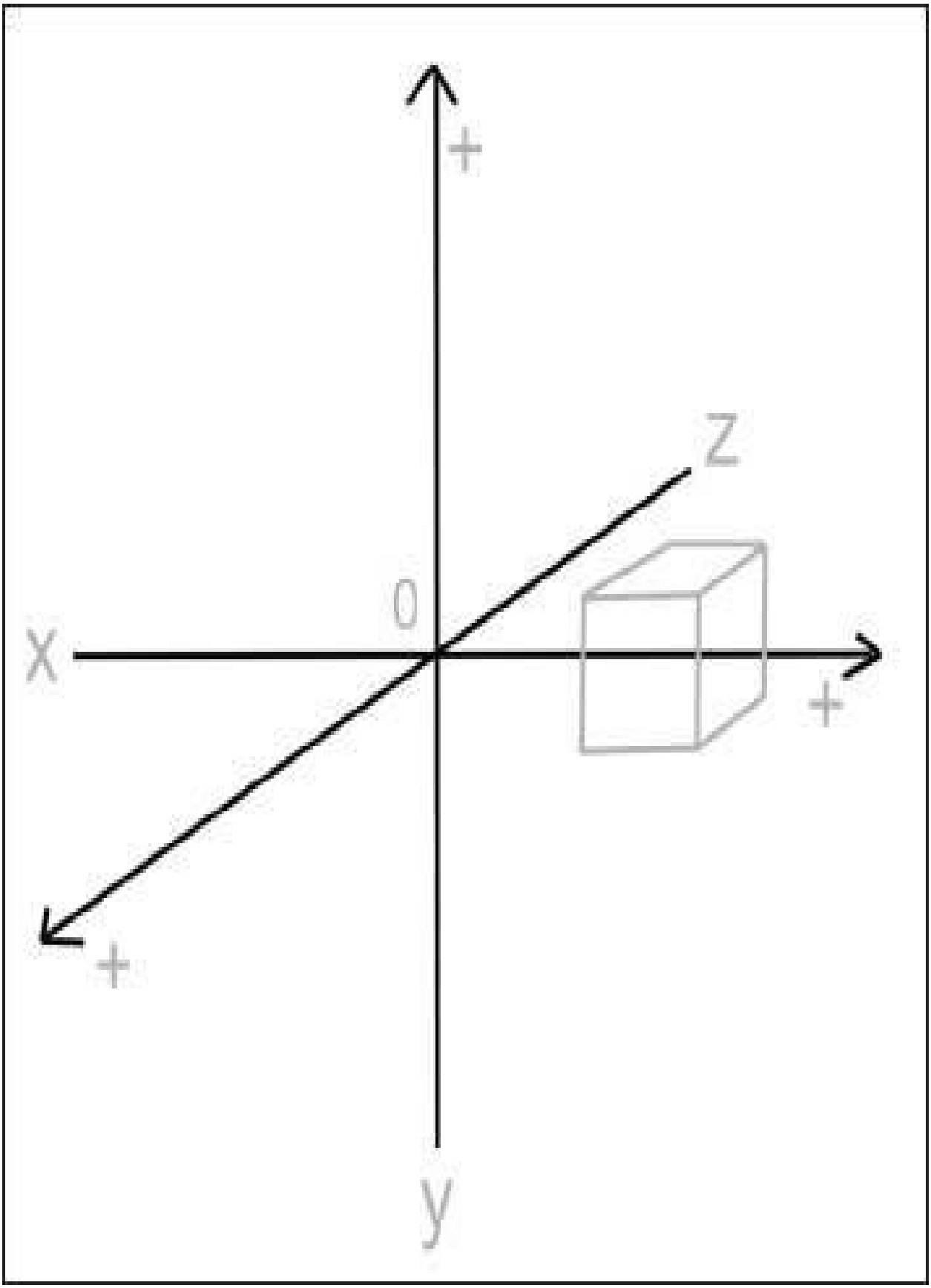

+
Z
0
X
+
+
y

Figura 11-2

Mover un elemento en el eje x.

Hay que establecer para cada nuevo elemento, ya sea una cámara, una luz o un objeto físico, los valores de las tres coordenadas que definen su posición en la escena. Cuando estas coordenadas no son declaradas, el elemento se coloca en el origen (las coordenadas 0, 0, 0). Debido a que no hay ningún parámetro para determinar el tamaño o la escala de un mundo 3D estándar, las unidades no tienen valores específicos; los valores utilizados son solo números decimales y la escala se establece por la relación entre los elementos ya definidos.

11.2.3 Cámara

La cámara es una parte esencial de la escena 3D. Determina el punto de vista del espectador (el usuario) y ofrece la perspectiva necesaria para proporcionar a nuestro mundo 3D un aspecto real. Three.js incluye constructores para crear dos tipos de cámaras:

PerspectiveCamera(campo, aspecto, cerca, lejos): Este constructor devuelve un objeto que representa una cámara con proyección en perspectiva. El atributo campo determina el campo de visión vertical, aspecto declara la relación de aspecto y los atributos cerca y lejos limitan lo que ve la cámara (los puntos más cercanos y más lejanos).

OrthographicCamera(izquierda, derecha, arriba, abajo, cerca, lejos): Este constructor devuelve un objeto que representa una cámara con proyección ortográfica. Los atributos izquierda, derecha, arriba, y abajo declaran el plano del tronco correspondiente. Los atributos cerca y lejos limitan lo que ve la cámara (el punto más cercano y el más lejano).

Perspectiva y proyecciones ortográficas son solo diferentes maneras de proyectar el mundo tridimensional sobre una superficie bidimensional como un lienzo. La proyección en perspectiva es más realista en cuanto a la imitación del mundo real y es el recomendado para las animaciones, pero la proyección ortográfica es útil para la visualización de las estructuras y dibujos que requieren acceso a los detalles, porque no tiene en cuenta algunos de los efectos producidos por el ojo humano.

El objeto Camera devuelto por estos constructores incluye un método para establecer el vector hacia el que tiene que apuntar la cámara:

lookAt(vector): Este método permite orientar la cámara hacia un punto específico en la escena. El atributo vector es un objeto vector que contiene las propiedades de los valores de las tres coordenadas x, y, y z (por ejemplo, {x: 10, y: 10, z: 20}). Por defecto, la cámara mira hacia el origen (las coordenadas 0, 0, 0).

11.2.4 Mallas

En un mundo 3D, los objetos físicos se representan a través de mallas. Una malla es una colección de vértices que definen una forma. Cada vértice de la malla es un nodo en el espacio tridimensional. Los nodos están unidos por líneas invisibles que generan pequeñas caras todo alrededor de la forma. El grupo de caras obtenido por la intersección de todos los vértices de una malla que constituye la superficie de la forma.

La Figura 11-3 muestra una malla que representa una esfera. Para crear esta particular malla tuvimos que definir 256 vértices que generaron 225 caras. Eso significa que tuvimos que declarar las coordenadas x, y, y z 256 veces para crear todos los nodos necesarios para la definición de una sola esfera. Este ejemplo demuestra lo difícil que es definir las formas tridimensionales, incluso cuando se trata de formas básicas como ésta. Debido a esta complejidad, los diseñadores trabajan con aplicaciones de modelado para generar gráficos 3D. Blender (www.blender.org), por ejemplo, exporta toda la malla en formatos especiales que pueden leer otras aplicaciones. Más adelante profundizaremos en ello.

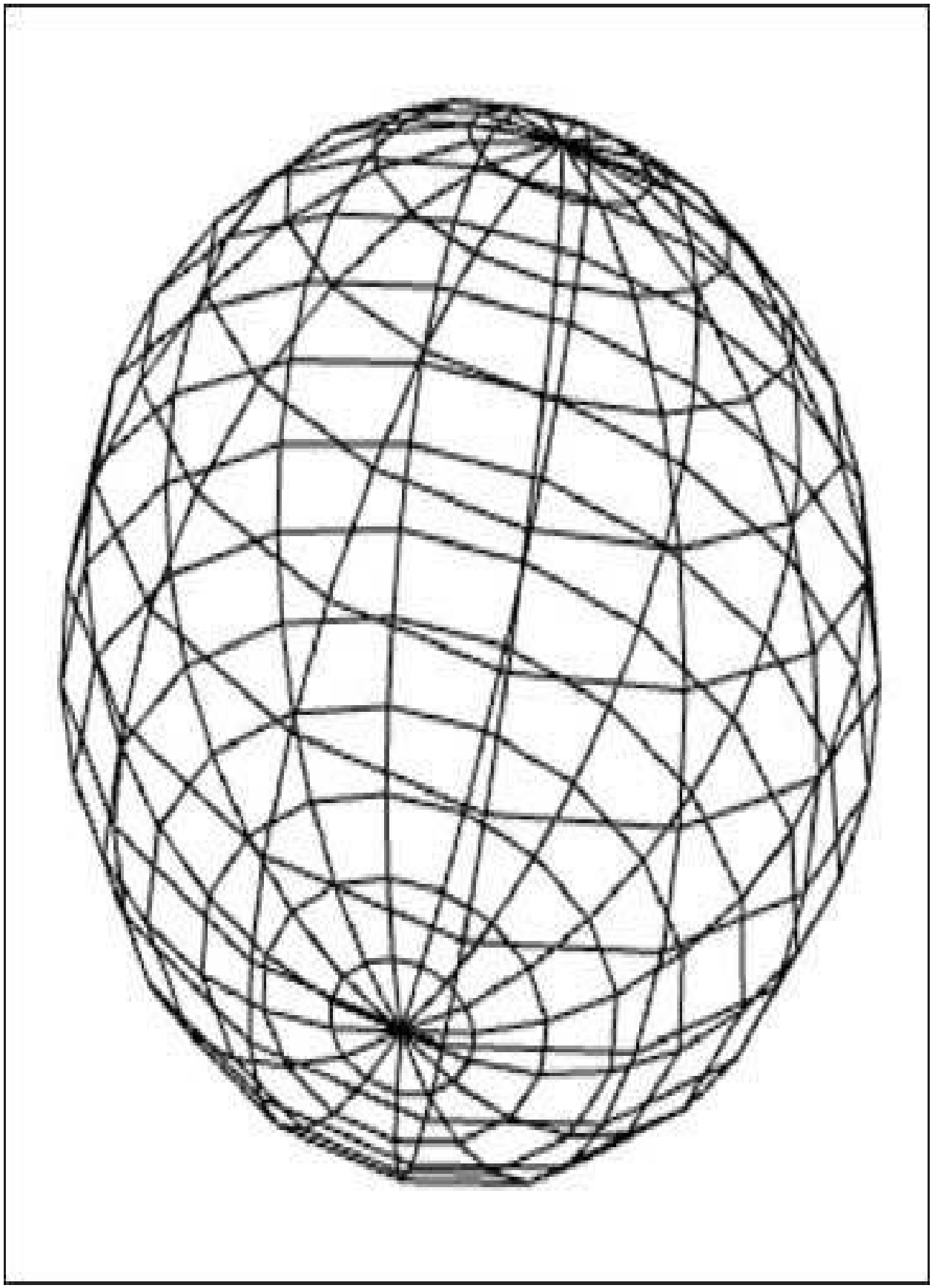


Figura 11-3

Malla para construir una esfera.

Una malla, como cualquier otro elemento, tiene que ser encerrada en un objeto antes de poder ser introducida en la escena. La biblioteca proporciona un método general para este propósito:

mesh(geometry, material): Este constructor devuelve un objeto que representa la malla. Los atributos geometry y material son objetos devueltos por constructores de geometría y materiales, como veremos a continuación.

11.2.5 Geométricas primitivas

Una esfera como la que se muestra en la Figura 11-3 es considerada una forma geométrica primitiva. Se trata simplemente de formas geométricas básicas que tienen una estructura definida, pero que se pueden modificar a través de determinados parámetros, como tamaño, dimensiones, etc. Para algunas aplicaciones, incluyendo juegos de vídeo pequeños, las formas geométricas primitivas son muy útiles y simplifican el trabajo de los diseñadores y desarrolladores. Three.js ofrece varios constructores para crear las formas primitivas geométricas más comunes:

SphereGeometry(radio, segmentos_horizontales, segmentos_verticales, inicioFi, largoFi, inicioZeta, largoTheta): Este constructor devuelve un objeto que contiene una malla para crear una esfera. El atributo radio define el radio de la esfera, los atributos segmentos_horizontales y segmentos_verticales declaran el número de segmentos incluidos horizontal y verticalmente para crear la forma, y la combinación de inicioFi, largoFi, inicioZeta y largoTheta nos permiten generar esferas incompletas. La mayoría de los atributos de este constructor son opcionales y tienen valores por defecto.

CubeGeometry(ancho, alto, profundidad, segmentos_horizontales, segmentos_verticales, profundidad_segmentos, materiales, lados): Este constructor devuelve un objeto que contiene una malla para construir un cubo. Los atributos ancho, ancho y profundidad determinan el tamaño de cada lado del cubo, los atributos segmentos_horizontales, segmentos_verticales y profundidad_segmentos determinan el número de segmentos a utilizar para crear las caras del cubo; el atributo materiales es una matriz que contiene materiales diferentes para cada cara, y el atributo lados contiene seis valores booleanos para especificar si se generará cada cara. La mayoría de los atributos son opcionales y tienen valores por

defecto.

CylinderGeometry(radio_superior, radio_inferior, altura, segmentos_horizontales, segmentos_verticales): Este constructor devuelve un objeto que contiene una malla para construir un cilindro. Los atributos radio_superior y radio_inferior especifican el radio de la parte superior e inferior del cilindro (diferentes radios construyen un cono); el atributo altura declara la altura del cilindro y los atributos segmentos_horizontales y segmentos_verticales declaran el número de segmentos horizontales y verticales que se utilizan para crear la malla.

IcosahedronGeometry (radio, detalle): Este constructor devuelve un objeto que contiene una malla para construir un icosaedro. El atributo radio declara el radio, y detalle especifica el nivel de detalle. Los niveles de detalle más altos requieren el uso de más segmentos. Usualmente, el valor de este atributo va desde 0 a 5, en función del tamaño de la forma y el nivel de detalle necesario.

OctahedronGeometry(radio, detalle): Este constructor devuelve un objeto que contiene una malla para construir un octaedro. El atributo radio declara la radio, y detalle especifica el nivel de detalle. Los niveles más altos requieren el uso de más segmentos. Usualmente el valor de este atributo va desde 0 a 5, en función del tamaño de la forma y el nivel de detalle necesario.

TetrahedronGeometry(radio, detalle): Este constructor devuelve un objeto que contiene una malla para construir un tetraedro. El atributo radio declara la radio y detalle especifica el nivel de detalle. Los niveles de detalle más altos requieren la utilización de más segmentos. Usualmente, el valor de este atributo va desde 0 a 5, en función del tamaño de la forma y del nivel de detalle necesario.

PlaneGeometry(ancho, alto, segmentos_horizontales, segmentos_verticales): Este constructor devuelve un objeto que contiene la malla para crear una superficie plana. Los atributos ancho y alto declaran la anchura y la altura correspondiente al plano, mientras que segmentos_horizontales y segmentos_verticales especifican cuántos segmentos se utilizarán para construirlo.

CircleGeometry(radio, segmentos, angulo_inicio, angulo_final): Este constructor devuelve un objeto que contiene una malla para construir un círculo plano. El atributo radio declara la radio del círculo, el atributo segmentos especifica el número de segmentos a utilizar para construir la forma y los atributos angulo_inicio y angulo_final declaran los ángulos en radianes en los que el círculo comienza y termina (para un punto de partida, los valores deben ser 0 y Math.PI * 2).

11.2.6 Materiales

WebGL utiliza shaders para representar gráficos en 3D en la pantalla. Los shaders son códigos programados en lenguaje GLSL (OpenGL Shading Language) que trabajan directamente con la GPU para producir la imagen en la pantalla. Proporcionan el nivel adecuado de luz y oscuridad en cada píxel de la imagen para generar la percepción de tres dimensiones. Este complejo concepto está escondido tras los materiales y luces de Three.js. Al definir materiales y luces, Three.js determina cómo se mostrará el mundo en 3D en la pantalla usando shaders preprogramados que son de aplicación común en animaciones 3D.

Three.js define unos pocos tipos de materiales que tienen que aplicarse de acuerdo a los requisitos de la aplicación y los recursos disponibles. Los materiales más realistas requieren más capacidad de procesamiento. El material elegido para su proyecto tendrá que equilibrar el nivel de realismo que requiere su aplicación y la capacidad de procesamiento disponible.

Es posible aplicar diferentes materiales a diferentes objetos del mismo mundo. Los siguientes constructores están disponibles para su definición:

LineBasicMaterial(parámetros): Este material se utiliza para representar mallas compuestas por líneas individuales (por ejemplo, una cuadrícula). El atributo parámetros es un objeto que contiene las propiedades de la configuración del material. Las propiedades disponibles son color (valor hexadecimal), opacity (valor decimal), blending (constante), depthTest (valor booleano) linewidth (valor decimal), LineCap (valores butt, round o square), Line-Join (valores round, bevel o miter), vertexColors (valor booleano) y fog (valor booleano).

MeshBasicMaterial(parámetros): Este material representa a la malla con un solo color, sin emular la reflexión de las luces. No es realista, pero útil en algunas circunstancias. El atributo parámetros es un objeto que contiene las propiedades de la configuración del material. Las propiedades disponibles son color (valor hexadecimal), opacity (valor decimal), map (objeto Texture), lightmap (objeto Texture), specularMap (objeto Texture), EnvMap (objeto TextureCube), combine (constante), reflectivity (valor decimal), refractionRatio (valor decimal), shading (constante), blending (constante), depthTest (valor booleano), wireframe (valor booleano) wireframelinewidth (valor decimal), vertexColors (constante), skinning (valor booleano) morphTargets (valor booleano) y fog (valor booleano).

MeshNormalMaterial(parámetros): Este material define un tono de color para cada cara de la malla. El efecto no es realista, porque las caras se distinguen una de otra, pero es particularmente útil cuando el hardware necesario para el cálculo de un material más realista no está disponible. El atributo parámetros es un objeto que contiene las propiedades de la configuración del material. Las propiedades disponibles son opacity (valor decimal), shading (constante), fog (constante), depthTest (valor booleano), wireframe (valor booleano) y wireframelinewidth (valor decimal).

MeshLambertMaterial(parámetros): Este material genera un sombreado suave en la superficie de la malla, produciendo el efecto de reflexión de la luz. El efecto es independiente del punto de la de vista del observador. El atributo parámetros es un objeto que contiene las propiedades de la configuración del material. Las propiedades disponibles son color (valor hexadecimal), ambient (valor hexadecimal), emissive (valor hexadecimal), opacity (valor decimal), map (objeto Texture), lightmap (objeto Texture), specularMap (objeto Texture), EnvMap (objeto TextureCube), combine (constante), reflectivity (valor decimal), refractionRatio (valor decimal), shading (constante), blending (constante), depthTest (valor booleano), wireframe (valor booleano), wireframelinewidth (valor decimal), vertexColors (constante), skinning (valor booleano), morphTargets (valor

booleano), morphNormals (valor booleano) y fog (valor booleano).

MeshPhongMaterial(parámetros): Este material produce un efecto realista con un sombreado suave sobre toda la superficie de la malla. El atributo parámetros es un objeto que contiene las propiedades de la configuración del material. Las propiedades disponibles son color (valor hexadecimal), ambient (valor hexadecimal), emissive, (valor hexadecimal), specular (valor hexadecimal), shininess (objeto Texture valor decimal), opacity (valor decimal), map (objeto Texture), lightmap (objeto Texture), bumpmap (objeto Texture), bumpScale (valor decimal), normalMap (objeto Texture), normalScale (objeto vectorial), specularMap (objeto Texture), EnvMap (objeto TextureCube), combine (constante), reflectivity (valor decimal), refractionRatio (valor decimal), shading (constante), blending (constante), depthTest (valor booleano), wireframe (valor booleano), wireframelinewidth (valor decimal), vertexColors (constante), skinning (valor booleano), morphTargets, (valor booleano), morphNormals (valor booleano) y fog (valor booleano).

MeshFaceMaterial(): Este constructor se aplica cuando se declaran diferentes materiales y texturas para cada cara de la geometría.

ParticleBasicMaterial(parámetros): Este material es específico para representar partículas (como humo, explosiones, etc.). El atributo parámetros es un objeto que contiene las propiedades de configuración del material. Las propiedades disponibles son color (valor hexadecimal), opacity (valor decimal), map (objeto Texture), size (valor decimal), sizeAttenuation (valor booleano) blending (constante), depthTest (valor booleano), vertexColors (valor booleano) y fog (valor booleano).

ShaderMaterial(parámetros): Este constructor nos permite ofrecer nuestros propios shaders. El atributo parámetros es un objeto que contiene las propiedades de configuración del material. Las propiedades disponibles

son: fragmentShader (cadena), vertexShader (cadena), uniforms (objeto), defines (objeto), shading (constante), blending (constante), depthTest (valor booleano), wireframe (valor booleano), wireframelinewidth (valor decimal), lights (valor booleano), vertexColors (constante), skinning (valor booleano), morphTargets (valor booleano) morphNormals, (valor booleano) y fog (valor booleano).

Importante

La mayoría de los parámetros de los constructores de materiales tienen valores predeterminados que son lo que los desarrolladores y diseñadores por lo general requieren. Vamos a mostrar cómo configurar algunos de ellos, pero para una referencia completa, consulte la documentación oficial y los ejemplos del sitio www.threejs.org.

Los constructores de materiales que hemos estudiado líneas atrás comparten propiedades comunes que se pueden añadir como parámetros para una configuración básica.

name: Esta propiedad define el nombre del material. Se establece una cadena vacía de forma predeterminada.

opacity: Define la opacidad del material. Toma valores de 0,0 a 1. El valor 1 (totalmente opaco) es aplicado por defecto.

transparent: Es una propiedad booleana que define la transparencia del

material. El valor por defecto es false.

visible: Define si el material es visible o no. El valor por defecto es true.

side: Esta propiedad define qué lado de la malla será representado. Puede usar tres constantes: THREE.FrontSide, THREE.BackSide o THREE.Double-Side.

11.2.7 Implementación

Toda esta teoría puede resultar desalentadora, pero de inmediato verá que la aplicación es muy simple. Vamos a ver un ejemplo de cómo crear una esfera como la de la Figura 11-3. Utilizaremos un elemento <canvas> de 500 × 400 píxeles, una cámara de perspectiva y un material básico para representar la malla en la pantalla.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Three.js</title>
  <script src="three.min.js"></script>
  <script src="webgl.js"></script>
</head>
<body>
  <section>
    <canvas id="canvas" width="500" height="400"></canvas>
  </section>
</body>
</html>
```

Código 11-1

Incluir la biblioteca Three.js en el documento HTML.

El paso más importante en cualquier aplicación Three.js es cargar la biblioteca. En el Código 11-1, el archivo three.min.js se incluye con uno de los elementos <script> del documento. El segundo <script> es para nuestro propio código Javascript.

```
function iniciar(){
  var lienzo = document.getElementById('lienzo');
  var anchura = lienzo.width;
  var altura = lienzo.height;

  var renderer = new THREE.WebGLRenderer({canvas: lienzo });
  var escena = new THREE.Scene();
  var camara = new THREE.PerspectiveCamera(75, anchura / altura,
 0.1, 1000);
  camara.position.set(0, 0, 150);

  var geometria = new THREE.SphereGeometry(80, 15, 15);
  var material = new THREE.MeshBasicMaterial({color: 0x000000,
 wireframe: true});
  var malla = new THREE.Mesh(geometria, material);
  escena.add(malla);

  renderer.render(escena, camara);
}
addEventListener('load', iniciar);
```

Código 11-2

Crear una esfera de alambre.

Como siempre, nuestra secuencia de comandos incluye la función iniciar() para comenzar a ejecutar la aplicación. Con esta función realizamos todo el proceso de creación del mundo 3D: configuramos el procesador, la escena, la cámara y una malla. En primer lugar, el procesador es configurado por el constructor WebGLRenderer(). Observe que todas las propiedades, métodos y constructores que ofrece Three.js forman parte del objeto de THREE. El constructor WebGLRenderer() es llamado desde este objeto, como se muestra en el Código 11-2 (THREE. WebGLRenderer()). El constructor recibe la propiedad canvas para establecer el elemento <canvas> como la superficie de representación y devuelve un objeto renderer que representará el procesador.

A continuación, se crea la escena mediante el constructor Scene(). No se necesitan parámetros para este constructor. El objeto Scene devuelto, se almacena en la variable escena para su uso posterior.

Una vez que tenemos la escena, el siguiente paso es crear la cámara. El uso del constructor PerspectiveCamera() permite conseguir una cámara con proyección en perspectiva (la recomendada para animaciones). El primer atributo declara una anchura para el punto de vista de 75. Este valor es adecuado para nuestra escena pero puede ser cambiado de acuerdo a la escala del mundo 3D. El segundo parámetro establece que la proporción de la cámara sea igual a la superficie de representación. Para obtener este valor, se divide la anchura entre la altura del lienzo, utilizando las variables altura y anchura definidas al inicio del código. Por último, se establece el límite más cercano en 0,1 y el más lejano en 1000. Estos valores también dependen de la escala que se utiliza para el resto de los elementos. Como los objetos no tendrán un tamaño mayor que 100 o 150

unidades, el límite de 1000 es suficiente para esta pequeña escena. Los objetos que vayan más lejos de este límite no serán dibujados en la pantalla. La cámara, como cualquier otro elemento del mundo, está situada inicialmente en el origen (las coordenadas 0, 0, 0). Para ser capaz de visualizar la malla creada a continuación, la cámara tiene que ser movida a una nueva posición. Esto se hace mediante la propiedad position y el método set(). Más adelante volveremos a ello. Por ahora, todo lo que necesita saber es que el método set() establece las coordenadas de la cámara como 0 para x, 0 para y y 150 para z, lo que efectivamente desplaza la cámara 150 unidades en el eje z.

Lo último que tenemos que añadir a nuestra escena es la malla. La malla y el material correspondiente son definidos por los constructores SphereGeometry() y MeshBasicMaterial() y luego son usados como atributos del constructor Mesh() para obtener el objeto final. El material se define con la propiedad wireframe establecida como true, para obtener una esfera de malla ven la pantalla en lugar de un objeto sólido. La malla se añade finalmente a la escena por el método add(), y la escena se representa en el lienzo al final de la función iniciar() con el método render() del objeto renderer.

Importante

Los colores en esta biblioteca están representados por números hexadecimales utilizando el prefijo 0x (Por ejemplo, 0xFF00FF).

Hágalo usted mismo

Vaya a www.threejs.org y descargue el paquete de la biblioteca. Extraiga los

archivos y copie el archivo three.min.js de la carpeta build en su carpeta de trabajo. Éste es el archivo que tiene que incluir en sus proyectos. Cree un nuevo archivo HTML con el documento del Código 11-1 y un archivo Javascript denominado webgl.js con el Código 11-2. Abra el documento en su navegador. El resultado debe ser similar a la Figura 11-3.

11.2.8 Transformaciones

Cada elemento del mundo 3D, incluyendo cámaras y luces, tiene atributos asignados por defecto. Se encuentran en el origen de la escena (las coordenadas 0, 0, 0), y su tamaño y orientación se determinan por las coordenadas de cada uno de sus vértices.

Es fácil modificar cámaras y luces pero las mallas requieren un nuevo cálculo de cada vértice. Para evitar una tarea de consumo de recursos tan elevado, Three.js ofrece un conjunto de propiedades y métodos para llevar a cabo las transformaciones más comunes. Utilizando una serie de propiedades que podemos trasladar, rotar o escalar un elemento sin tener que volver a calcular sus miles de vértices.

position: Esta propiedad devuelve un objeto que contiene un vértice que permite obtener o establecer la posición de un elemento. El objeto proporciona las propiedades x, y y z para leer o cambiar cada coordenada de forma independiente.

rotation: Esta propiedad devuelve un objeto que contiene un vértice que permite obtener o establecer el ángulo del elemento en radianes. El objeto proporciona las propiedades x, y y z para leer o cambiar el ángulo de cada coordenada de forma independiente.

scale: Esta propiedad devuelve un objeto que contiene un vértice que permite obtener o establecer la escala de un elemento. El objeto proporciona las propiedades x, y y z para leer o cambiar la escala de cada coordenada de

forma independiente. Por defecto se establece un valor igual a 1.

Por lo general, los valores de las tres coordenadas tienen que ser modificados al mismo tiempo. Three.js proporciona el método set() para simplificar este proceso. El método es aplicable a todas las propiedades de esta transformación y los valores se declaran separados por una coma, como se muestra en la secuencia de comandos del Código 11-2.

```
var renderer, escena, camara, malla;
function iniciar(){
  var lienzo = document.getElementById('lienzo');
  var anchura = lienzo.width;
  var altura = lienzo.height;

  renderer = new THREE.WebGLRenderer({canvas: lienzo });
 escena = new THREE.Scene();
  camara = new THREE.PerspectiveCamera(45, anchura / altura, 0.1,
 1000);
  camara.position.set(0, 0, 150);

var geometria = new THREE.CubeGeometry(50, 50, 50);
  var material = new THREE.MeshBasicMaterial({color: 0x000000,
 wireframe: true});
  malla = new THREE.Mesh(geometria, material);
  escena.add(malla);

  lienzo.addEventListener('mousemove', mover);
}

function mover(e){
  malla.rotation.x = e.pageY * 0.01;
  malla.rotation.z = -e.pageX * 0.01;
  renderer.render(escena, camara);
}

addEventListener('load', iniciar);
```

Código 11-3

Rotación de un cubo con el ratón.

El ejemplo del Código 11-3 muestra cómo transformar dinámicamente una malla. Para una mejor visualización, generamos un cubo con el constructor CubeGeometry(). El resto de la secuencia es similar a la del ejemplo anterior, pero al final de la función iniciar() se ha añadido un detector para el evento mousemove para crear una animación simple. Ahora, cada vez que el puntero del ratón se mueve sobre el elemento <canvas>, se llama a la función mover() y la malla gira en los ejes x y z según la posición del puntero del ratón.

Hágalo usted mismo

Copie el texto del Código 11-3 en el archivo webgl.js creado en el ejemplo anterior y abra el documento del Código 11-1 en su navegador.

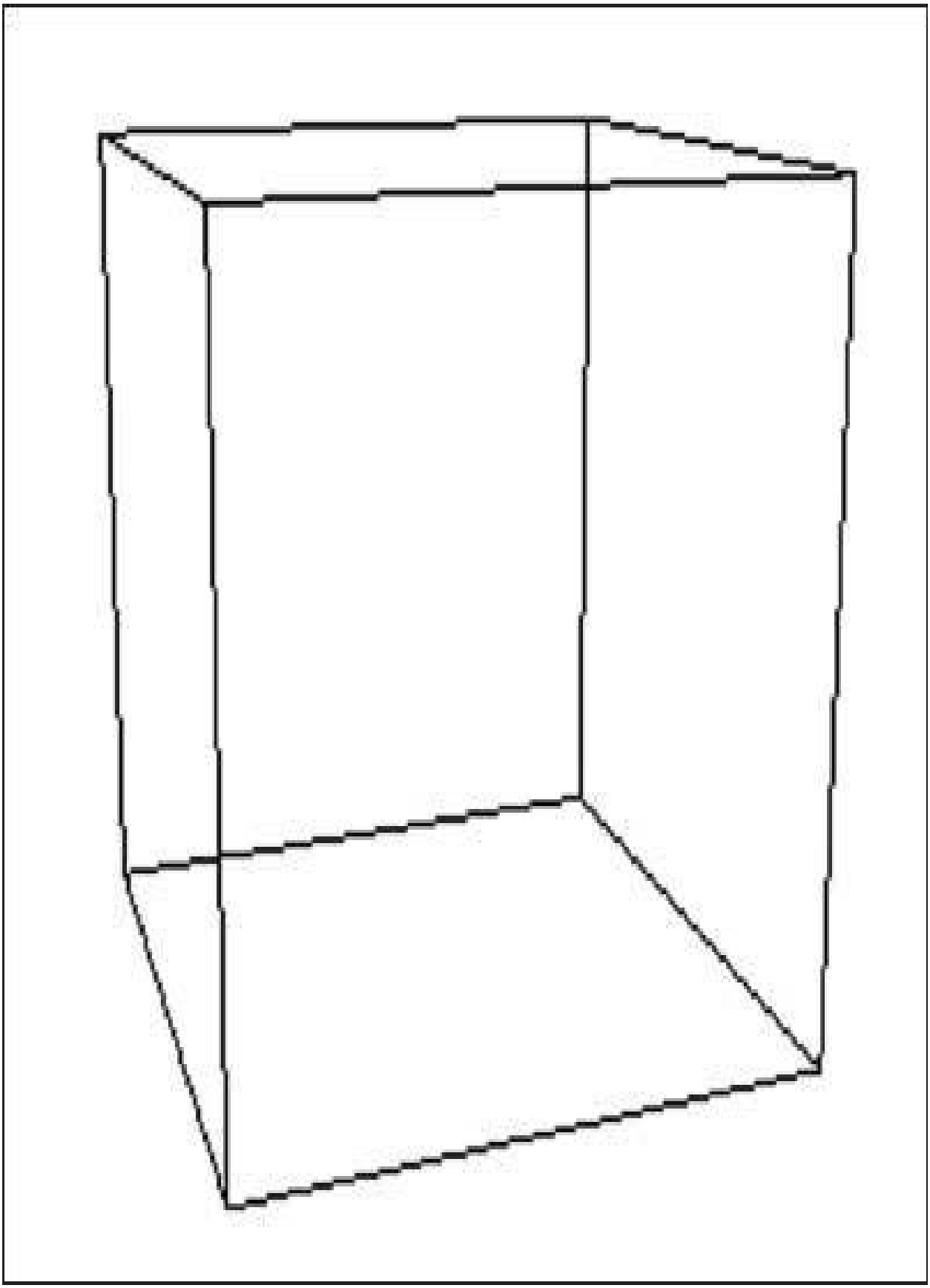


Figura 11-4

Cubo de alambre animado.

11.2.9 Luces

Para crear mallas con materiales que pueden imitar la reflexión de la luz, necesitamos algo de luz. Three.js varios constructores para generar luz.

AmbientLight(color): Ésta es una luz global que no se atribuye a ninguna fuente de luz específica. Está formada por una luz general que se refleja en todos los objetos de la escena y afecta a todos los objetos por igual. El atributo color es el color de la luz expresando en un valor hexadecimal.

DirectionalLight (color, intensidad, distancia): Este tipo de luz se encuentra muy lejos del espacio 3D y afecta a los objetos a partir de una sola dirección. Los atributos posibles son color (valor hexadecimal), intensidad (valor decimal) y distancia (valor decimal).

PointLight (color, intensidad, distancia): Este constructor crea una fuente de luz en una ubicación específica en el espacio 3D. Afecta a los objetos en todas direcciones. Los atributos posibles son color (valor hexadecimal), intensidad (valor decimal) y distancia (valor decimal).

SpotLight (color, intensidad, distancia, sombraproyectada): Este constructor crea una fuente de luz en una ubicación específica y con una dirección específica. Los atributos posibles son color (valor hexadecimal), intensidad (valor decimal), distancia (valor decimal) y sombraproyectada (valor booleano).

```
function iniciar(){
  var canvas = document.getElementById('lienzo');
  var anchura = lienzo.width;
  var altura = lienzo.height;

  var renderer = new THREE.WebGLRenderer({canvas: lienzo, antialias:
 true});
  var escena = new THREE.Scene();
  var camara = new THREE.PerspectiveCamera(45, anchura / altura,
 0.1, 1000);
  camara.position.set(0, 0, 150);

  var geometria = new THREE.CubeGeometry(50, 50, 50);
  var material = new THREE.MeshPhongMaterial({color: 0x0000FF});
  var malla = new THREE.Mesh(geometria, material);
  escena.add(malla);

  malla.rotation.set(10, 10, 0);

  var luz = new THREE.SpotLight(0xFFFFFF);
  luz.position.set(50, 50, 150);
  escena.add(luz);

  renderer.render(escena, camara);
}
addEventListener('load', iniciar);
```

Código 11-4

Adición de luz a nuestra escena.

Además de la fuente de luz añadida al final, hay algunos cambios más en el Código 11-4. La configuración del procesador incluye ahora una segunda propiedad llamada antialias con el valor true. Esto suaviza la imagen en la pantalla, creando un efecto más realista. Esta malla del cubo es construida por el constructor CubeGeometry() y el material se define como MeshPhongMaterial(), de color azul. El material Phong es probablemente el más realista, pero también el más exigente. Es perfecto para nuestro ejemplo, pero siempre se debe considerar la posibilidad de usar materiales alternativos para conseguir un equilibrio entre el rendimiento y la calidad.

Hágalo usted mismo

Copie las órdenes Javascript del Código 11-4 en el archivo webgl.js creado en el ejemplo anterior y abra el documento HTML que contiene el Código 11-1 en su navegador.

El cubo se crea por defecto en el origen del espacio 3D (las coordenadas 0, 0, 0) y se enfrenta a la cámara desde un lateral. Si procesamos la malla en esta posición, solo seremos capaces de ver un cuadrado en la pantalla. Para hacer más atractivo el cubo, hemos aplicado una transformación mediante la propiedad rotation y el método set().

Para la luz se utilizó el constructor SpotLight() con color blanco. Debido a que esta luz tiene un lugar específico, tenemos que moverla a la posición correcta. Esto se hace con la propiedad position y el método set().

Figura 11-5

Luz reflejada en un cubo.

11.2.10 Texturas

Los colores y la reflexión de la luz producen un efecto muy realista, pero el nivel de detalle de la realidad es mucho más que eso. La reproducción de la vida real en una pantalla de ordenador no es tan fácil como tomar una fotografía con una cámara. Las protuberancias, las partículas, los tejidos e incluso las pequeñas moléculas de una superficie puede producir millones de efectos visuales. Los detalles en el mundo real suelen ser tan complejos que reproducir un objeto realista requeriría recursos no disponibles en los ordenadores hoy en día. Para resolver este problema, los motores 3D ofrecen la posibilidad de añadir texturas a las mallas. Las texturas son imágenes que se dibujan en la superficie de una forma para simular detalles complejos. Las texturas en Three.js son creadas por un constructor específico y luego son declaradas como parte del material de la malla. La biblioteca ofrece el constructor Texture() para este propósito lo siguiente:

Texture(imagen, mapeo, envolturaS, envolturaT, filtroMag, filtroMin): Este constructor devuelve un objeto que representa la textura que se aplicará al material para la malla. El atributo imagen es la imagen de la textura. Puede ser declarada como una imagen, un elemento del lienzo o un vídeo. El atributo mapeo define cómo se aplica la textura a la superficie de la malla; los atributos wraps y wrapT determinan cómo se distribuye la imagen sobre la superficie, y los atributos filtroMag y filtroMin configuran los filtros para aplicar a la textura con el fin de suavizar la imagen y producir un efecto más realista.

```
var lienzo, imagen, renderer, escena, camara, malla;
function iniciar(){
  lienzo = document.getElementById('lienzo');

  imagen = document.createElement('img');
  imagen.setAttribute('src', 'crate.jpg');
  imagen.addEventListener('load', createworld);
}
function createworld(){
  var width = lienzo.width;
  var height = lienzo.height;

  renderer = new THREE.WebGLRenderer({canvas: lienzo, antialias:
 true});
  escena = new THREE.Scene();
  camara = new THREE.PerspectiveCamera(45, width / height, 0.1,
 1000);
  camara.position.set(0, 0, 150);

  var geometry = new THREE.CubeGeometry(50, 50, 50);
  var textura = new THREE.Texture(imagen);
  textura.needsUpdate = true;
  var material = new THREE.MeshPhongMaterial({map: textura});
  malla = new THREE.Mesh(geometry, material);
  escena.add(malla);

  var light = new THREE.SpotLight(0xFFFFFF, 1);
  light.position.set(0, 100, 250);
  escena.add(light);

  lienzo.addEventListener('mousemove', mover);
}
function mover(e){
  malla.rotation.z = -e.pageX * 0.01;
  malla.rotation.x = e.pageY * 0.01;
  renderer.render(escena, camara);
}
addEventListener('load', iniciar);
```

Código 11-5

Añadir texturas a nuestro objeto.

El archivo que contiene la imagen de la textura tiene que ser completamente descargado desde el servidor antes de que pueda ser aplicada al material. En la mayoría de las aplicaciones, el código está diseñado para descargar los recursos e indicar el progreso al usuario. En el ejemplo del Código 11-5 separaramos este proceso desde la creación del espacio 3D para ofrecer una perspectiva de cómo debe ser organizado el código. Más adelante en este capítulo vamos a estudiar mejores alternativas.

La función iniciar() crea un objeto Image, declara el archivo crate.jpg como fuente de la imagen y añade la función createworld() como detector para el evento load con el fin de continuar el proceso cuando el archivo se haya cargado.

La función createworld() sigue el mismo procedimiento que se ha utilizado antes de crear el espacio 3D y todos sus elementos, pero en este ejemplo la textura se define antes de aplicar el material a la malla. El objeto Texture fue generado por el constructor Texture() con la referencia a la imagen previamente descargada. Una vez que la textura está lista, se almacena en la variable textura y se asigna al material como uno de sus atributos. Todo este proceso se suele denominar “asignación de texturas”, y la propiedad a cargo de la asignación de la textura al material que se conoce como map.

Hágalo usted mismo

Copie las órdenes Javascript del Código 11-5 en el archivo webgl.js, suba el archivo, la imagen de la textura y el documento del Código 11-1 para el servidor y abra el documento HTML en su navegador. La imagen de este ejemplo está disponible en www.minkbooks.com/content/.

Importante

Debido a las restricciones en cuanto al uso de orígenes múltiples, los archivos de las texturas tienen que estar ubicados en el mismo servidor que la aplicación. Visite www.minkbooks.com/content/ y descargue los recursos para este libro.

FRAGILE

Figura 11-6

Nuestro cubo simple ahora se ve como una caja de madera.

El objeto Texture ofrece algunas propiedades para ayudar a definir y configurar la textura:

needsUpdate: Esta propiedad informa al procesador de que es necesaria una actualización para la textura. Es necesario cada vez que se define una nueva textura o se hacen cambios en una textura existente (ver ejemplo del Código 11-5).

repeat: Esta propiedad define cuántas veces tiene que ser repetida la imagen de la textura en el mismo lado de la malla. Devuelve o establece un vértice con dos coordenadas, x e y (para cada eje de la cara). Esta propiedad requiere los atributos envolturaS y envolturaT del constructor Texture que se establecerá como THREE.RepeatWrapping, y la imagen utilizada para la textura tiene que ser de una potencia de 2 (por ejemplo, 128 × 128, 256 × 256, 512 × 512, 1024 × 1024, etc).

offset: Esta propiedad desplaza la imagen a la superficie, lo que permite, entre otros efectos, crear una animación simple pero eficaz. Devuelve y establece un vértice con dos coordenadas, x e y (para cada eje de la cara). Esta propiedad requiere los atributos envolturaS y envolturaT del constructor de textura que se establecerá como THREE.RepeatWrapping, y la imagen utilizada para la textura tiene que ser de una potencia de 2 (por ejemplo, 128 × 128, 256 × 256, 512 × 512, 1024 × 1024, etc.).

11.2.11 Aplicación UV

Aplicación UV es un proceso en el que los puntos específicos de la imagen de la textura están asociados con los vértices de la malla para dibujar la imagen en una posición exacta. El nombre deriva de los nombres asignados a los ejes de la imagen: U corresponde al eje × de la textura y V corresponde al eje y, pues los nombres x e y ya se utilizan para describir los ejes de los vértices de la malla.

Afortunadamente, Three.js se ocupa de la mayoría de los cálculos y proporciona alternativas más sencillas para la aplicación de texturas en la mayoría de circunstancias comunes. Para formas primitivas geométricas básicas, la biblioteca está configurada con los valores predeterminados para distribuir la imagen de la forma esperada, como podemos ver en la caja de madera en el último ejemplo. Esta lógica preprogramada nos permite componer la imagen de acuerdo con la forma de la malla a la que se desea aplicar y deja el resto del trabajo para la biblioteca.

Un efecto importante logrado por el mapeado UV es la aplicación de diferentes materiales, texturas y por tanto, para una misma geometría. El procedimiento está generalmente reservado para mallas complejas creadas por software de 3D, pero en el caso de la geometría del cubo, Three.js proporciona una manera fácil de hacerlo:

```
var renderer, scene, camera, mesh;
function iniciar(){
  canvas = document.getElementById('canvas');
  var width = canvas.width;
  var height = canvas.height;

  renderer = new THREE.WebGLRenderer({canvas: canvas, antialias:
                        true});
  scene = new THREE.Scene();
  camera = new THREE.PerspectiveCamera(45, width / height, 0.1,
                        1000);
  camera.position.set(0, 0, 150);

  var materials = [new THREE.MeshPhongMaterial({map: THREE
                        .ImageUtils.loadTexture('dice3.jpg')}),
    new THREE.MeshPhongMaterial({map: THREE
                        .ImageUtils.loadTexture('dice4.jpg')}),
    new THREE.MeshPhongMaterial({map: THREE
                        .ImageUtils.loadTexture('dice5.jpg')}),
    new THREE.MeshPhongMaterial({map: THREE
                        .ImageUtils.loadTexture('dice2.jpg')}),
    new THREE.MeshPhongMaterial({map: THREE
                        .ImageUtils.loadTexture('dice1.jpg')}),
    new THREE.MeshPhongMaterial({map: THREE
                        .ImageUtils.loadTexture('dice6.jpg')})
  ];
  var geometry = new THREE.CubeGeometry(50, 50, 50, 1, 1, 1,
                        materials);
  mesh = new THREE.Mesh(geometry, new THREE.MeshFaceMaterial());
  scene.add(mesh);

  var light = new THREE.SpotLight(0xFFFFFF, 2);
  light.position.set(0, 100, 250);
  scene.add(light);

  canvas.addEventListener('mousemove', move);
}
function move(e){
  mesh.rotation.x = e.pageY * 0.01;
  mesh.rotation.z = : e.pageX * 0.01;
  renderer.render(scene, camera);
}
addEventListener('load', iniciar);
```

Código 11-6

Uso de una imagen diferente para cada lado del cubo.

En este ejemplo se usa el método loadTexture() del objeto ImageUtils proporcionado por la biblioteca para descargar los archivos y asignar las texturas al material sin alterar el resto del proceso. Las mallas y el espacio se generan en 3D y las texturas son aplicadas más tarde, cuando las imágenes se han descargado. Ésta es una manera fácil de trabajar con texturas en aplicaciones sencillas. El método loadTexture() descarga la imagen y devuelve el objeto Texture correspondiente, todo en un paso. Nos ahorra así la creación de un código para descargar los recursos, pero como no proporciona un buen control del proceso, no se recomienda para el desarrollo profesional; solo lo hemos incluido el método para simplificar el ejemplo.

Hágalo usted mismo

Copie las órdenes Javascript del Código 11-6 en el archivo webgl.js, suba el archivo, las imágenes y el documento del Código 11-1 a su servidor y abra el documento HTML en su navegador. Visite www.minkbooks.com/content/ para descargar las imágenes de este ejemplo.

Mediante loadTexture() y el constructor MeshPhongMaterial(), creamos una matriz con seis objetos Material diferentes, cada uno con su textura correspondiente. Esta matriz es luego asignada como un atributo al constructor CubeGeometry(). Debido a que todas las definiciones ya han sido formuladas

por este constructor, el constructor Mesh() solo requiere el objeto Geometry, pero también tenemos que utilizar el constructor MeshFaceMaterial() como segundo atributo para indicar los tipos de materiales incluidos por la geometría. Éste es un constructor que devuelve un tipo de material compuesto para la aplicación de materiales diferentes a la misma malla.

El orden de los materiales y texturas declaradas en la matriz son importantes pues determinan la ubicación exacta en la que las imágenes se dibujan en la superficie de la malla.

Importante

Aplicar textura a formas complejas requiere un conocimiento profundo del Mapeo UV y del uso de texturas en general. Para obtener más información, por favor visite nuestro sitio web y siga los enlaces de este capítulo.

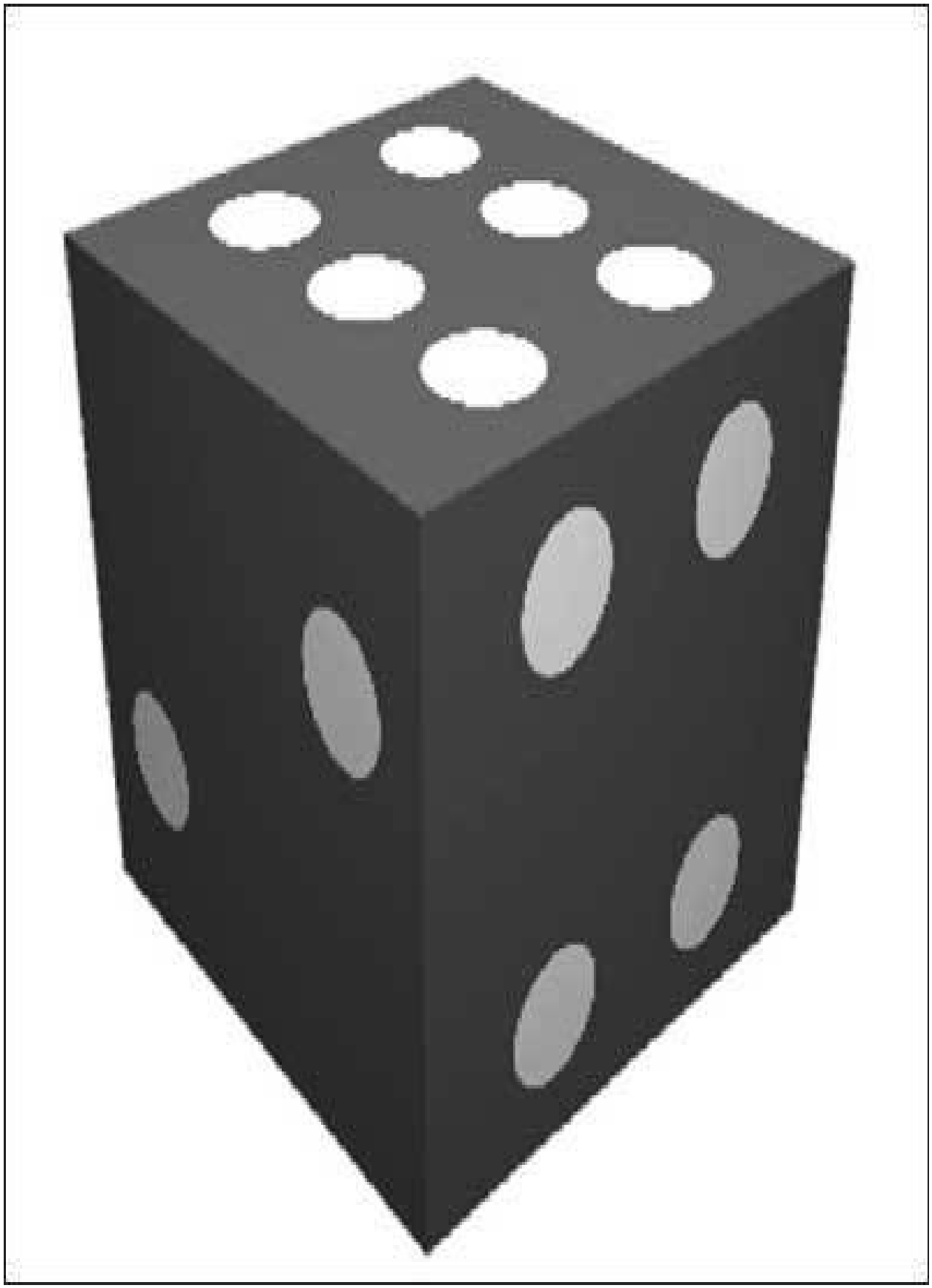


Figura 11-7

Hacer el cubo en un troquel.

11.2.12 Texturas de lienzo

Existen diferentes combinaciones de técnicas que se pueden aplicar para crear efectos impresionantes utilizando texturas, pero probablemente uno de los más interesantes es el uso de un elemento <canvas> adicional. Esta alternativa nos da acceso a una variedad de métodos que proporciona la API Canvas para generar dinámicamente una textura. Pero una de las cosas más geniales sobre el uso de un elemento <canvas> como la fuente de la textura es lo fácil que es incluir texto en 3D en la escena.

```
var renderer, escena, camara, malla;
function iniciar(){
  lienzo = document.getElementById('lienzo');
  var ancho = lienzo.width;
  var alto = lienzo.height;
  renderer = new THREE.WebGLRenderer({canvas: lienzo, antialias:true});
  renderer.setClearColorHex(0xCCFFFF);
  escena = new THREE.Scene();
  camara = new THREE.PerspectiveCamera(45, ancho / alto, 0.1, 1000);
  camara.position.set(0, 0, 150);
  var textcanvas = document.createElement('canvas');
  textcanvas.setAttribute('width', '400');
  textcanvas.setAttribute('height', '200');
  var context = textcanvas.getContext('2d');
  context.fillStyle = "rgba(255,0,0,0.95)";
  context.font = "bold 70px verdana, sans-serif";
  context.fillText("Text in 3D", 0, 60);
  var geometria = new THREE.PlaneGeometry(100, 40);
  var textura = new THREE.Texture(textcanvas);
  textura.needsUpdate = true;
  var material = new THREE.MeshPhongMaterial({map: textura, side:
  THREE.DoubleSide});
  malla = new THREE.Mesh(geometria, material);
  escena.add(malla);
  var luz = new THREE.PointLight(0xffffff);
  luz.position.set(0, 100, 250);
  escena.add(luz);
  lienzo.addEventListener('mousemove', move);
}
  function move(e){
  malla.rotation.y = e.pageX * 0.02;
  renderer.render(escena, camara);
}
addEventListener('load', iniciar);
```

Código 11-7

Incluir texto en 3D en la escena.

El lienzo de la textura es un segundo elemento <canvas> que no se muestra en la pantalla. Su único propósito es el de generar la imagen de la textura. Podríamos haber declarado este elemento en el documento HTML y utilizar CSS para cambiar su propiedad visibility, pero es preferible crear el objeto para el DOM (Document Object Model) de forma dinámica con el método createElement() y trabajar con este objeto en su lugar.

Hágalo usted mismo

Copie las órdenes Javascript del Código 11-7 en el archivo webgl.js y abra el documento HTML que contiene el Código 11-1 en su navegador. Mueva el puntero del ratón sobre el lienzo para rotar la malla y ver la textura de ambos lados.

En el Código 11-7, crea el segundo elemento <canvas> y define sus atributos width y height. El proceso de elaboración de un texto en el lienzo es el mismo que se explicó en el capítulo anterior: se crea el contexto 2D, se define el estilo del texto y el texto se dibuja usando el método fillText().

Una de las ventajas del uso del elemento <canvas> como fuente para la textura

es su capacidad para hacer transparentes las partes de la superficie de dibujo que no están en uso. Para demostrar este efecto, declaramos un color de fondo para la representación con el método setClearColorHex() y aplicamos la textura a una geometría plana. Ésta es una malla de plana con solo dos lados, como un cuadrado tridimensional. La geometría se crea usando el constructor PlaneGeometry().

El resto del proceso es el mismo, pero se añade un nuevo parámetro al constructor MeshPhongMaterial() para declarar el material de doble cara. Esto significa que la textura se muestra en el frente y la parte posterior de la malla, produciendo un efecto de pantalla de cine.

Texto en 3D

Figura 11-8

Texto 3D en una geometría plana.

11.2.13 Texturas de vídeo

Hablando de cine, ¿qué opina sobre usar un vídeo como el origen de la textura? Es casi tan simple como aplicar la referencia a un elemento <video> como atributo del constructor Texture().

```
var lienzo, video, renderer, escena, camara, malla;
function iniciar(){
  lienzo = document.getElementById('lienzo');
  video = document.createElement('video');
  video.setAttribute('src', 'trailer.ogg');
  video.addEventListener('canplaythrough', createworld);
}
function createworld(){
  var ancho = lienzo.width;
  var alto = lienzo.height;
  renderer = new THREE.WebGLRenderer({canvas: lienzo,
 antialias:true});
  escena = new THREE.Scene();
  camara = new THREE.PerspectiveCamera( 45, ancho / alto, 0.1,
 1000);
  camara.position.set(0, 0, 250);

  textura = new THREE.Texture(video);
  textura.minFilter = THREE.LinearFilter;
  textura.magFilter = THREE.LinearFilter;
  textura.generateMipmaps = false;
  var material = new THREE.MeshPhongMaterial({map: textura, side:
                       THREE.DoubleSide});
  var geometry = new THREE.PlaneGeometry(240, 135);
  malla = new THREE.Mesh(geometry, material);
  escena.add(malla);

  var luz = new THREE.PointLight(0xffffff);
  luz.position.set(0, 100, 250);
  escena.add(luz);

  lienzo.addEventListener('mousemove', move);
  video.play();
  render();
}
function move(e){
  malla.rotation.y = e.pageX * 0.02;
}
function render(){
  textura.needsUpdate = true;
  renderer.render(escena, camara);
  webkitRequestAnimationFrame(render);
}
addEventListener('load', iniciar);
```

Código 11-8

Presentación de vídeo en un mundo 3D.

Utilizamos la expresión “casi tan simple” en la descripción de este procedimiento, ya que, como se puede ver en el Código 11-8, para reproducir un vídeo en la superficie de una malla se requiere alguna configuración y también la creación de un bucle para actualizar la textura en cada fotograma de vídeo.

El código comienza con la creación del elemento <video>. La fuente para el elemento se declara como trailer.ogg, el mismo archivo utilizado en los ejemplos del Capítulo 6. Para saber cuándo está listo para ser reproducido el vídeo, se añade un detector al evento canplaythrough. Una vez que el navegador considera que hay suficientes datos para iniciar la reproducción del vídeo, el evento se dispara y la función createworld() se ejecuta.

Importante

El código presentado usa el método requestAnimationFrame() para ejecutar el bucle, pero en nuestro ejemplo se declara para la versión experimental de Google Chrome. Para ejecutar esta aplicación en un navegador diferente, tiene que declarar el método usando el prefijo correspondiente. Esto ya no será necesario una vez que esta API se convierta en estándar.

El procedimiento para la creación del espacio 3D es similar al de los ejemplos anteriores, pero esta vez tenemos que configurar el objeto Textura para poder usar el video como textura. De forma predeterminada, Three.js aplica un filtro a las texturas mipmap, que produce un efecto antialias para suavizar la textura y hacer que parezca más realista, pero las texturas de vídeo no son compatibles con este filtro. Después de la creación del objeto Texture, establecemos las propiedades minFilter y magFilter a THREE.LinearFilter (un filtro simple) y desactivamos el filtro mipmaps.

Sin embargo, nuestro trabajo no termina ahí. Todavía es necesario crear el bucle para actualizar la textura en cada cuadro y representar la escena. Esto se hace con la función render(). En esta función, la propiedad needsUpdate de la textura se establece como true, y la escena es nuevamente representada.

Al final de la función createworld (), se añade un detector del evento mousemove para proporcionar algún tipo de interacción con la malla. También se inicia la reproducción del vídeo con método play() y se llama a la función render() por primera vez para iniciar el bucle de procesamiento.

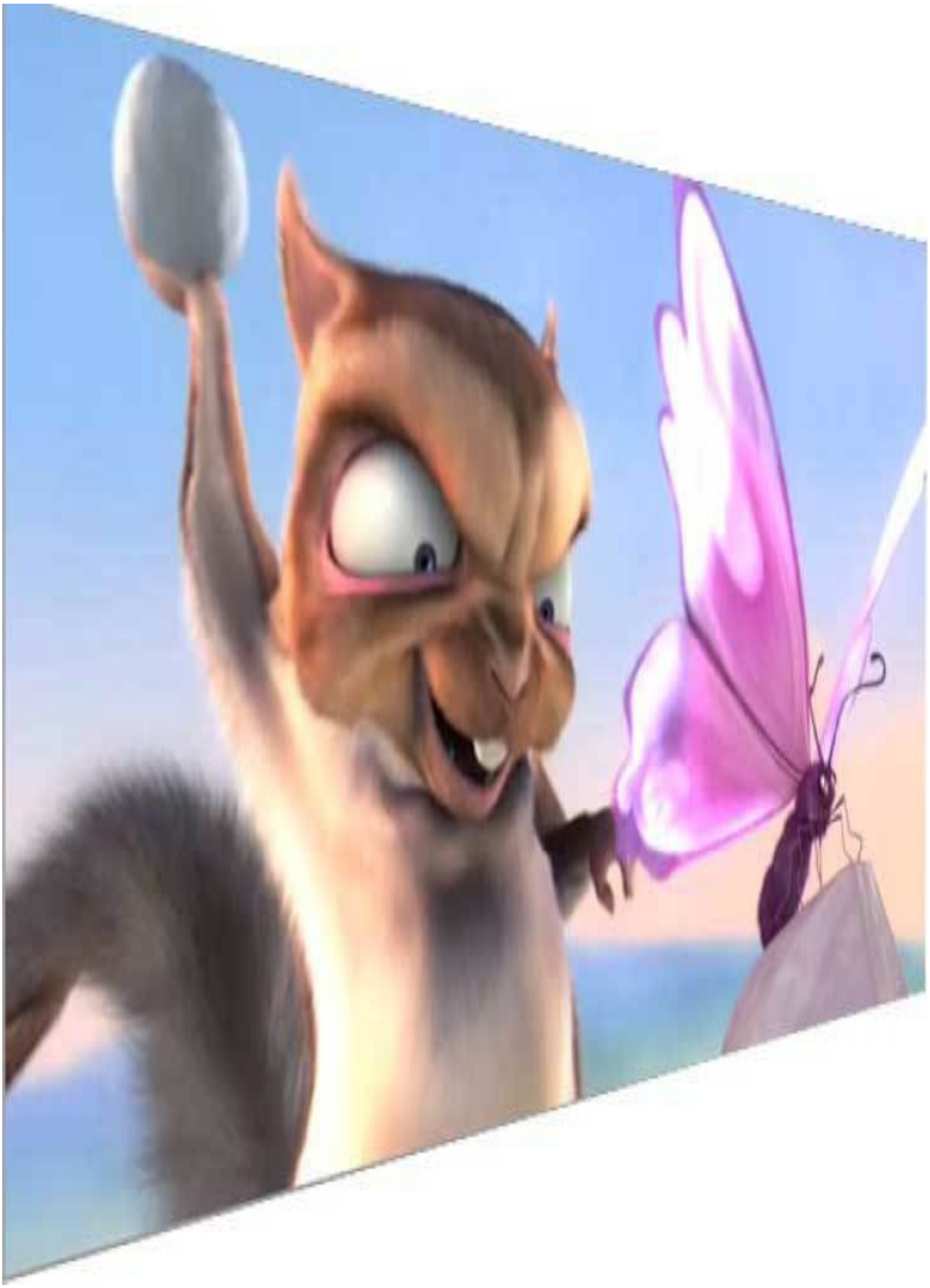


Figura 11-9 Vídeo textura. © Copyright 2008, Blender Foundation www.bigbuckbunny.org

Hágalo usted mismo

Copie las órdenes Javascript del Código 11-8 en el archivo webgl.js, suba el archivo, el vídeo y el documento HTML con el Código 11-1 al servidor y abra el documento HTML en su navegador. Visite www.minkbooks.com/content/ para descargar el vídeo para este ejemplo.

Importante

Un vídeo OGG solo se reproduce en los navegadores que soportan el formato OGG, como Google Chrome, Mozilla Firefox u Opera. Consulte el Capítulo 6 para obtener más información.

11.2.14 Cargar modelos 3D

Es casi imposible desarrollar mallas complejas declarando los vértices individualmente o mediante los constructores de geométricas primitivas básicas. Hacerlo exige el uso de software 3D profesional capaz de construir modelos elaborados que luego pueden ser cargados y ejecutados por las aplicaciones en 3D. Uno de los programas más populares en el mercado es Blender (www.blender.org), creado por la Fundación Blender y distribuido de forma gratuita. Este programa ofrece todas las herramientas necesarias para crear modelos 3D profesionales y también varios formatos en los que los modelos pueden ser exportados. Los desarrolladores y los colaboradores de la biblioteca Three.js han construido códigos externos para ayudar a los programadores de cargar archivos en estos formatos y adaptarlos de manera que Three.js los pueda entender y procesar.

Ya hay cargadores disponibles para los formatos COLLADA, OBJ, UTF8 y JSON, entre otros. Los cargadores son archivos Javascript que tienen que ser incluidos en el documento junto con el archivo Three.js. Veamos un ejemplo usando el formato Collada y el archivo ColladaLoader.js:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Three.js</title>
  <script src="three.min.js"></script>
  <script src="ColladaLoader.js"></script>
  <script>
    var canvas, video, renderer, escena , camara, malla;
    function iniciar(){
      lienzo = document.getElementById('lienzo');
      var loader = new THREE.ColladaLoader();
      loader.load('police.dae', createworld);
    }
    function createworld(collada){
      var width = lienzo.width;
      var height = lienzo.height;

      renderer = new THREE.WebGLRenderer({canvas: lienzo,
                      antialias:true});
      escena  = new THREE.Scene();
      camara = new THREE.PerspectiveCamera(45, width / height, 0.1,
                      1000);
      camara.position.set(0, 0, 150);

      malla = collada.scene;
      malla.scale.set(20, 20, 20);
      malla.rotation.set(-Math.PI / 2, 0, 0);
      escena .add(malla);
      var light = new THREE.PointLight(0xffff);
      light.position.set(0, 100, 250);
      escena .add(light);

      lienzo.addEventListener('mousemove', move);
    }


    function move(e){
      malla.rotation.z = -e.pageX * 0.01;
      renderer.render(escena , camara);
    }
    addEventListener('load', iniciar);
  </script>
</head>
<body>
  <section>
    <canvas id="lienzo" width="500" height="400"></canvas>
  </section>
</body>
</html>
```

Código 11-9

Cargar modelos 3D.

Los modelos exportados como COLLADA se almacenan en un grupo de archivos. Básicamente, después de exportar el modelo vamos a tener un archivo de texto con la extensión .dae que contiene todas las especificaciones del modelo y archivos de imágenes para las texturas. El modelo de nuestro ejemplo se almacena en el archivo police.dae, y son creados dos archivos para las texturas (SFERIFF.JPG y SFERIFFI.JPG). Vamos a cargar este modelo para agregar un viejo coche de policía a la escena.

Los archivos que contienen los modelos tienen que ser descargados como cualquiera de los recursos utilizados anteriormente, pero el cargador de COLLADA ofrece su propio método para hacerlo. En la función iniciar() del Código 11-9, el constructor ColladaLoader() se utiliza para crear el objeto Loader, y el método load() proporcionado por el objeto llama a la descarga al archivo .dae (solo tiene que ser declarado en el método el archivo .dae, los archivos de las texturas se descargan de forma automática). El método load() tiene dos atributos, el atributo file para indicar la ruta del archivo a descargar y el atributo callback para declarar una función que se llama cuando el archivo se ha descargado. El método load() envía un objeto Scene a esta función que puede procesar para insertar el modelo de nuestra propia escena.

En nuestro ejemplo, la función que procesa el objeto Scene es createworld(). Después de seguir el procedimiento estándar de crear el procesador (renderer), la escena y la cámara, el objeto Scene que representa el modelo se almacena en la variable malla, es escalado a las dimensiones del espacio 3D, rotado hacia la cámara desde el ángulo derecho y, finalmente, añadido a la escena.

El modelo es ahora parte de nuestro mundo en 3D y es representado al mover el puntero del ratón sobre el elemento <canvas>.

Figura 11-10

Modelo COLLADA Proporcionado por TurboSquid Inc. (www.turbosquid.com).

El archivo para el cargador de COLLADA está disponible en en la carpeta examples del paquete Three.js. Vaya a www.threejs.org y descargue el paquete de la biblioteca si aún no lo ha hecho. Luego siga la ruta examples/js/ loaders/ y copie el archivo ColladaLoader.js a la carpeta de su proyecto. Este archivo tiene que ser incluido en el documento como lo hicimos en el Código 11-9 para tener acceso a los métodos que nos permitan leer y procesar archivos en este formato. Cree un nuevo archivo HTML con el texto del Código 11-9, cargue este archivo, el documento HTML y los tres archivos para el modelo en su servidor y abra el documento en su navegador.

Importante

El modelo utilizado en este ejemplo viene acompañado de dos ficheros que contienen las imágenes para las texturas (SFERIFF.JPG y SFERIFFI.JPG). Los tres archivos de este modelo están disponibles en www.minkbooks.com/content/.

11.2.15 Animaciones en 3D

Las animaciones en 3D solo se diferencian de las animaciones en 2D por la forma en la que se construyen los cuadros. En WebGL, el procesador hace la mayoría del trabajo duro y todo el proceso parece ser casi automático, mientras que en el contexto de un lienzo 2D, tenemos que tener cuidado de limpiar nuestra propia superficie. A pesar de las pequeñas diferencias, los requisitos de los códigos para una aplicación profesional son siempre los mismos. Cuanto mayor es la complejidad, mayor es la necesidad de una organización adecuada, y la mejor manera de establecer esta organización es trabajar declarando propiedades y métodos dentro de un objeto global común.

Para dar un ejemplo de la animación 3D, vamos a crear un pequeño videojuego. La organización del código Javascript va a ser similar a la usada para la animación 2D en el Capítulo 10. En el juego vamos a conducir un coche en torno a un gran espacio rodeado por murallas. El objetivo es capturar las esferas verdes que están flotando alrededor. Vamos a empezar con la construcción del documento HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Three.js</title>
  <style>
    body{
      margin: 0px;
      overflow: hidden;
    }
  </style>
  <script src="three.min.js"></script>
  <script src="ColladaLoader.js"></script>
  <script src="webgl.js"></script>
</head>
<body></body>
</html>
```

Código 11-10

Documento HTML para nuestro videojuego en 3D.

El documento HTML es extremadamente simple. Los estilos CSS para el elemento <body> pueden parecer extraños. Como vamos a utilizar la ventana completa del navegador para el procesador, estas propiedades son necesarias para asegurarnos de que ni márgenes ni barras de desplazamiento roban espacio a nuestra aplicación. Los archivos de Javascript incluidos son el archivo three.min.js de la biblioteca Three.js, el archivo ColladaLoader.js para traer a escena de nuevo a nuestro fantástico coche de policía y el archivo webgl.js para la propia aplicación.

Hágalo usted mismo

Cree un archivo para el texto del Código 11-10 con un nombre y la extensión .html. Copie el código Javascript que se presenta en esta parte del capítulo en un archivo vacío llamado webgl.js. Todos los scripts estudiados a partir de ahora son necesarios para ejecutar la aplicación.

Conceptos básicos

La propiedad overflow es una propiedad CSS tradicional que determina cómo

responde el navegador cuando el contenido excede los límites de su caja padre. Puede tomar cuatro valores posibles: visible, hidden, scroll o auto. En nuestro ejemplo hemos asignado el valor hidden para ocultar el contenido más allá de los límites del cuerpo y evitar que las barras de desplazamiento se muestren automáticamente cuando hay más contenido disponible fuera de los límites del elemento.

El código Javascript para una aplicación 3D es generalmente más amplio que el necesario para otros fines. Para simplificar este ejemplo, vamos a presentar primero el objeto global con las propiedades elementales y luego agregaremos el resto de las propiedades y métodos requeridos para el juego en los códigos posteriores de este mismo capítulo.

```
  var mijuego = {
    renderer: '',
    scene: '',
    camara: '',
    luz: '',
    coche: {
      malla: '',
      speed: 0,
      speedUP: false,
      left: false,
      right: false,
      wheelangle: 0
    },
    paredes: [{x: 0, y: 100, z: -1000},
              {x: -1000, y: 100, z: 0},
              {x: 0, y: 100, z: 1000},
              {x: 1000, y: 100, z:0}],
    texturas: {
      coche: '',
      suelo: '',

    paredes: ''
  },
  targets: {
    malla: '',
    bordes: [],
  },
  input: []
}
```

Código 11-11

Definir propiedades básicas.

El objeto global de la aplicación se llama mijuego. Partimos de la definición de mijuego declarando propiedades necesarias para controlar y modificar el estado del juego. Las propiedades renderer, scene, camara y luz almacenan referencias a los elementos de nuestro espacio 3D. La propiedad coche contiene un objeto con datos básicos del coche, tales como una referencia a la malla y el valor de la velocidad actual. La propiedad paredes es una matriz que contiene cuatro vértices para la definición de la posición de las paredes. La propiedad texturas es un objeto con propiedades para hacer referencia a las texturas para el coche, el piso y las paredes. La propiedad targets constituye otro un objeto que almacena la malla y los bordes de las esferas verdes (los objetivos de nuestro juego). Y la propiedad input es una matriz vacía que almacena la entrada del usuario.

Algunas de estas propiedades se inicializan con valores nulos. El siguiente paso es declarar los valores adecuados y definir los elementos básicos de nuestra escena. Esto se hace usando el método iniciar().

```
mijuego.iniciar = function(){
  var ancho = window.innerWidth;
  var alto = window.innerHeight;
  var lienzo = document.createElement('canvas');
  lienzo.setAttribute('width', ancho);
  lienzo.setAttribute('height', alto);
  document.body.appendChild(lienzo);


  mijuego.renderer = new THREE.WebGLRenderer({canvas: lienzo,
                        antialias:true});
  mijuego.renderer.setClearColorHex(0x000000);
  mijuego.scene = new THREE.Scene();
  mijuego.camara = new THREE.PerspectiveCamera(45, ancho / alto,
                        0.1, 10000);
  mijuego.camara.position.set(0, 50, 150);

  mijuego.luz = new THREE.PointLight(0xFFFFFF);
  mijuego.luz.position.set(0, 50, 150);
  mijuego.scene.add(mijuego.luz);

  addEventListener('keydown', function(e){mijuego.input.push({type:
                        'keydown', key: e.keyCode});});
  addEventListener('keyup', function(e){mijuego.input.push({type:
                        'keyup', key: e.keyCode});});

  mijuego.loading();
  mijuego.create();
};
```

Código 11-12

Definición del método iniciar().

Queremos utilizar la ventana completa del navegador para nuestro juego, pero las dimensiones de este espacio siempre varían en función del dispositivo que ejecuta la aplicación, el navegador utilizado para cargar el documento e incluso el tamaño de la ventana actual fijado por el usuario o el sistema. Para crear un elemento <canvas> del tamaño de la ventana, primero tenemos que obtener las dimensiones de la ventana actual usando las propiedades innerWidth y innerHeight del objeto Window. Usando estos valores, la tela se crea dinámicamente, se ajusta su tamaño y se añade al cuerpo gracias al método appendChild().

Después de las definiciones habituales del procesador o renderer, la escena, la cámara y la luz, se agregan dos detectores para los eventos keydown y keyup. Estos detectores son necesarios para controlar la entrada del usuario. El evento keydown se activa cuando se pulsa una tecla, y el evento keyup se activa cuando se suelta una tecla. Ambos devuelven la propiedad keyCode que contiene un valor que identifica la tecla que generó el evento. Para detectar los eventos y almacenar los valores de keyCode en la matriz input cuando aquellos son disparados, se establecen funciones anónimas. Procesaremos esta entrada posteriormente.

Al final del método iniciar() se llama al método loading() para descargar los archivos con la descripción del modelo y de las texturas.

```
mijuego.loading = function(){
  var loader = new THREE.ColladaLoader();
  loader.load('police.dae', function(collada){ mijuego.texturas.
                              coche = collada; });

  var img = document.createElement('img');
  img.setAttribute('src', 'wet_asphalt.jpg');
  img.addEventListener('load', function(e){ mijuego.texturas.suelo =
                              e.target; });

  var img = document.createElement('img');
  img.setAttribute('src', 'wall.jpg');
  img.addEventListener('load', function(e){ mijuego.texturas.paredes
                              = e.target; });

  var controlloop = function(){
    if(mijuego.texturas.coche && mijuego.texturas.suelo && mijuego.
                              texturas.paredes){
      mijuego.create();
    }else{
      setTimeout(controlloop, 200);
    }
  };
  controlloop();
};
```

Código 11-13

Definición del método loading().

Este método es un código cargador pequeño pero práctico. Inicia el proceso de descarga para cada uno de los archivos (el modelo COLLADA, la textura para el suelo y la textura para las paredes) y define una pequeña función interna para comprobar la situación.

Cuando los archivos se han descargado por completo, los objetos resultantes se almacenan en las propiedades correspondientes (texturas.coche, texturas.suelo y texturas.paredes). La función controlloop() se llama a sí misma generando un bucle para comprobar constantemente el valor de estas propiedades y ejecutar el método create() solo cuando el modelo y las texturas se han cargado ya.

```
mijuego.create = function(){
  var geometry, material, texture, malla;
  malla = mijuego.texturas.coche.scene;
  malla.scale.set(20, 20, 20);
  malla.rotation.set(-Math.PI / 2, 0, Math.PI);
  malla.position.y += 14;
  mijuego.scene.add(malla);
  mijuego.coche.malla = malla;
  geometry = new THREE.PlaneGeometry(2000, 2000, 10, 10);
  texture = new THREE.Texture(mijuego.texturas.suelo, THREE.
        UVMapping, THREE.RepeatWrapping, THREE.RepeatWrapping);
  texture.repeat.set(20, 20);
  texture.needsUpdate = true;
  material = new THREE.MeshPhongMaterial({map: texture});
  malla = new THREE.Mesh(geometry, material);
  malla.rotation.x = Math.PI * 1.5;
  mijuego.scene.add(malla);

  for(var f = 0; f < 4; f++){
    geometry = new THREE.PlaneGeometry(2000, 200, 10, 10);
    texture = new THREE.Texture(mijuego.texturas.paredes, THREE.
        UVMapping, THREE.RepeatWrapping, THREE.RepeatWrapping);
    texture.repeat.set(10, 1);
    texture.needsUpdate = true;

    material = new THREE.MeshPhongMaterial({map: texture});
    malla = new THREE.Mesh(geometry, material);
    malla.position.set(mijuego.paredes[f].x, mijuego.paredes[f].y,
        mijuego.paredes[f].z);
    malla.rotation.y = Math.PI / 2 * f;
    mijuego.scene.add(malla);
  }
  mijuego.loop();
};
```

Código 11-14

Definición del método create().

El método create(), crea las mallas para el coche, el suelo y las cuatro paredes. No hay nada nuevo en este método, excepto el uso de la propiedad repeat. Esta propiedad declara cuántas veces se repite la imagen al lado de la malla para crear una textura. Por ejemplo, la imagen de la textura para las paredes es un cuadrado perfecto, pero las paredes son diez veces más largas que altas. Al establecer los valores de la propiedad repeat como 10, 1 (es decir, texture.repeat.set(10, 1);), la imagen se dibuja en la pared en las proporciones adecuadas.

Como hemos mencionado antes, para que la propiedad repeat funcione correctamente es necesario cumplir algunas condiciones. El tamaño de la imagen para la textura tiene que ser una potencia de 2 (por ejemplo, 128 × 128, 256 × 256, 512 × 512, 1024 × 1024, etc.); hay que declarar más de un segmento por el lado de la geometría plana y los atributos Wraps y wrapT del constructor Texture() tiene que ser ajustado a THREE.RepeatWrapping.

Los atributos mencionados se establecen de forma predeterminada para THREE.ClampToEdgeWrapping, lo que significa que la imagen se escala para ocupar el tamaño de todo el lado de la malla (como se ha visto en los ejemplos anteriores). La constante THREE.RepeatWrapping anula este efecto y permite distribuir la imagen de la forma deseada.

La definición del espacio 3D está lista y es el momento de programar el núcleo de la aplicación, es decir, los métodos que se encargarán del procesamiento principal. Necesitamos un método para controlar la entrada del usuario, otro

método para calcular nuevas posiciones para los elementos dinámicos, un tercer método para detectar las colisiones y uno más para actualizar el procesador.

```
mijuego.control = function(e){
  var action;
  while(mijuego.input.length){
    action = mijuego.input.shift();
    switch(action.type){
      case 'keydown':
        switch(action.key){
          case 38:
            mijuego.coche.speedUP = true;
            break;
          case 37:
            mijuego.coche.left = true;
            break;
          case 39:
            mijuego.coche.right = true;
            break;
        }
        break;
      case 'keyup':
        switch(action.key){
          case 38:
            mijuego.coche.speedUP = false;
            break;
          case 37:
            mijuego.coche.left = false;
            break;
          case 39:
            mijuego.coche.right = false;
            break;
        }
        break;
    }
  }
};
```

Código 11-15

Definición del método control().

El método control() del Código 11-15 procesa los valores almacenados en la entrada matriz por los detectores de los eventos keydown y keyup. Esto es parte de una técnica utilizada para evitar el retardo producido por el sistema cada vez que se pulsa una tecla. Cuando el usuario pulsa o suelta una tecla, la acción es detectada por estos eventos y la nueva condición almacena las propiedades speedUP, left o right de acuerdo con su naturaleza. Estas propiedades devolverán el valor true desde el momento en que se pulsa la tecla hasta que se libera. Usando este procedimiento no habrá ningún retraso y las teclas que controlan el coche responderán al instante.

Para identificar qué tecla ha pulsado o soltado el usuario, el valor keyCode tomado de la matriz input se compara con los números 38, 37 y 39. Estos códigos corresponden a las teclas Desplazamiento hacia arriba, Desplazamiento hacia la izquierda y Desplazamiento hacia la derecha respectivamente. De acuerdo con estos valores y el evento al que responden, las propiedades speedUP, left o right se establecen como true o false para indicar la condición actual del resto de los métodos del código.

Conceptos básicos

Cada tecla del teclado está asociada a un valor único. Usando estos valores podemos identificar la tecla exacta que ha sido presionada o soltada. Para obtener una lista completa de los valores, visite nuestro sitio web y siga los

enlaces de este capítulo.

Importante

Un sistema de entrada similar al programado para esta aplicación puede ser utilizado para almacenar cualquier tipo de entrada, no solamente las teclas. Podemos almacenar valores personalizados para los eventos de ratón en la matriz input y luego responder a los valores desde el método control() de la misma manera que hicimos en este ejemplo para las teclas de desplazamiento.

Las propiedades speedUP, left o right son extremadamente relevantes. Nos ayudan a definir la velocidad y la rotación del coche. Todo lo demás se calcula utilizando esta información, desde la posición de la cámara hasta la detección de colisiones contra las paredes o las esferas. Una gran parte de este trabajo es realizado por el método process().

```
mijuego.process = function(){
  if(mijuego.coche.speedUP){
    if(mijuego.coche.speed < 8){
      mijuego.coche.speed += 0.1;
    }
  }else{
    if(mijuego.coche.speed > 0){
      mijuego.coche.speed -= 0.1;
    }else{
      mijuego.coche.speed += 0.1;
    }
  }
  if(mijuego.coche.left && mijuego.coche.wheelangle > - 0.5){
    mijuego.coche.wheelangle -= 0.01;
  }
  if(!mijuego.coche.left && mijuego.coche.wheelangle < 0){
    mijuego.coche.wheelangle += 0.02;
  }
    if(mijuego.coche.right && mijuego.coche.wheelangle < 0.5){
      mijuego.coche.wheelangle += 0.01;
    }
    if(!mijuego.coche.right && mijuego.coche.wheelangle > 0){
      mijuego.coche.wheelangle -= 0.02;
    }

    var angle = mijuego.coche.malla.rotation.z;
    angle -= mijuego.coche.wheelangle * mijuego.coche.speed / 100;
    mijuego.coche.malla.rotation.z = angle;

    mijuego.coche.malla.position.x += Math.sin(angle) * mijuego.coche.
   speed;
    mijuego.coche.malla.position.z += Math.cos(angle) * mijuego.coche.
   speed;

    var deviation = mijuego.coche.wheelangle / 3;
    posx = mijuego.coche.malla.position.x - (Math.sin(mijuego.coche.
   malla.rotation.z + deviation) * 150);
    posz = mijuego.coche.malla.position.z - (Math.cos(mijuego.coche.
   malla.rotation.z + deviation) * 150);
    mijuego.camara.position.set(posx, 50, posz);
    mijuego.camara.lookAt(mijuego.coche.malla.position);
    mijuego.luz.position.set(posx, 50, posz);
  };
```

Código 11-16

Definición del método process().

Hay dos valores importantes que deben ser recalculados en cada ciclo del bucle para procesar la entrada del usuario: la velocidad y el ángulo del coche. La velocidad aumenta mientras el usuario mantiene presionada la tecla de desplazamiento hacia arriba y disminuye cuando ésta se libera. La primera declaración if else en el método process() del Código 11-16 controla la situación. Si el valor de SpeedUp es true, la velocidad del coche se incrementa en 0,1 hasta un máximo de 8. Si el valor es false, la velocidad se reduce gradualmente a 0.

Las siguientes cuatro declaraciones if controlan el ángulo de las ruedas. Este ángulo se añade o se resta al ángulo del coche para dirigirlo hacia la izquierda o la derecha de acuerdo con los valores de las propiedades left y right. Como sucedía con las declaraciones relacionadas con la velocidad, las declaraciones if para el ángulo de las ruedas establecen límites para el posible valor, que va de −0.5 a 0.5.

Una vez que los valores para la velocidad y el ángulo de las ruedas son establecidos, comienza el cálculo para determinar la nueva posición de cada elemento en la escena. En primer lugar, el ángulo actual del vehículo es almacenado en la variable angle (es el ángulo del eje z). A continuación, el ángulo de las ruedas es retado del valor del ángulo del coche, y este nuevo valor es asignado a su vez a la propiedad rotation del coche para girar la malla (mijuego.coche.malla.rotation.z = angle).

Con el ángulo ajustado, es hora de determinar la nueva posición del vehículo.

Los valores para las coordenadas x y z se calculan a partir del ángulo y la velocidad con la fórmula Math.sin(angle) × speed and Math.cos(angle) × speed. Los resultados son asignados inmediatamente a las propiedades position.x y position.z del coche, moviendo así efectivamente la malla a la nueva posición, pero todavía tenemos que trasladar la cámara y la luz a lo largo de los ejes o el vehículo pronto desaparecerá en las sombras.

Usando los valores para establecer la nueva posición del vehículo, el ángulo y la distancia permanente entre la cámara y el vehículo (150), se calculan los valores de las coordenadas de la cámara. Estos valores se almacenan en las variables posx y posy, y son asignados a la propiedad position de la cámara y de la luz (la cámara y la luz siempre tienen las mismas coordenadas). Con esta fórmula, la cámara se ve como si estuviera unida a la parte trasera del coche. Para lograr un efecto más realista, se calcula una pequeña desviación a partir del ángulo de las ruedas y se suma al ángulo del coche en la última fórmula, para mover así la cámara ligeramente a un lado u otro (deviation = mijuego.car.wheelangle / 3).

Al final del método process(), la cámara es dirigida hacia la nueva posición del coche con el método lookAt() y el vértice devuelto por la propiedad position (mijuego.coche.malla.position). No importa cuándo cambie el valor de esta propiedad, la cámara siempre apunta hacia el coche.

Conceptos básicos

Las fórmulas matemáticas aplicadas en este ejemplo son simples funciones trigonométricas utilizadas para obtener los valores de coordenadas desde ángulos y puntos específicos en un sistema de bidimensional de coordenadas. Para más ejemplos, vaya al Código 10-28, Capítulo 10.

Con las posiciones del coche, la cámara y la luz definidas, estamos listos para representar la escena. solo queda un elemento. Las esferas verdes que el coche tiene que perseguir se colocan al azar en toda la escena, de una en una. Cuando el coche pasa a través de la esfera actual, la malla desaparece y se crea una nueva en una posición. Como esta operación puede ocurrir en cualquier momento durante la animación, hemos decidido incluirla junto al proceso de representación en el método draw():

```
mijuego.draw = function(){
  if(!mijuego.targets.malla){
    var geometry = new THREE.SphereGeometry(20, 10, 10);
    var material = new THREE.MeshBasicMaterial({color: 0x00FF00,
 wireframe: true});
    var malla = new THREE.Mesh(geometry, material);
    var posx = (Math.random() * 1800) - 900;
    var posz = (Math.random() * 1800) - 900;
    malla.position.set(posx, 30, posz);
    mijuego.scene.add(malla);

    mijuego.targets.malla = malla;
    mijuego.targets.bordes[0] = posx - 30;
    mijuego.targets.bordes[1] = posx + 30;
    mijuego.targets.bordes[2] = posz - 30;
    mijuego.targets.bordes[3] = posz + 30;

  }else{
    mijuego.targets.malla.rotation.y += 0.02;
  }
  mijuego.renderer.render(mijuego.scene, mijuego.camara);
};
```

Código 11-7

Definición del método draw().

Si aún no ha sido creada ninguna esfera, el método draw() genera una nueva. El material se configura como una estructura de alambre verde, y la posición se determina aleatoriamente por el método random(). Después de que la malla se añade a la escena, se calcula un área alrededor de la esfera para poder detectar su posición más adelante.

La posición de cada elemento de la escena está determinada por un solo vértice. Sería casi imposible de detectar una colisión entre los elementos comparando solo estas coordenadas, pero sumando y restando 30 unidades a cada coordenada de la esfera y almacenando esos valores en la matriz targets.edges, se genera un espacio virtual de 60 unidades de ancho que contiene la esfera (ver Figura 11-11). Cuando el vértice de la posición del coche se encuentra dentro de esta zona, la colisión se confirma.

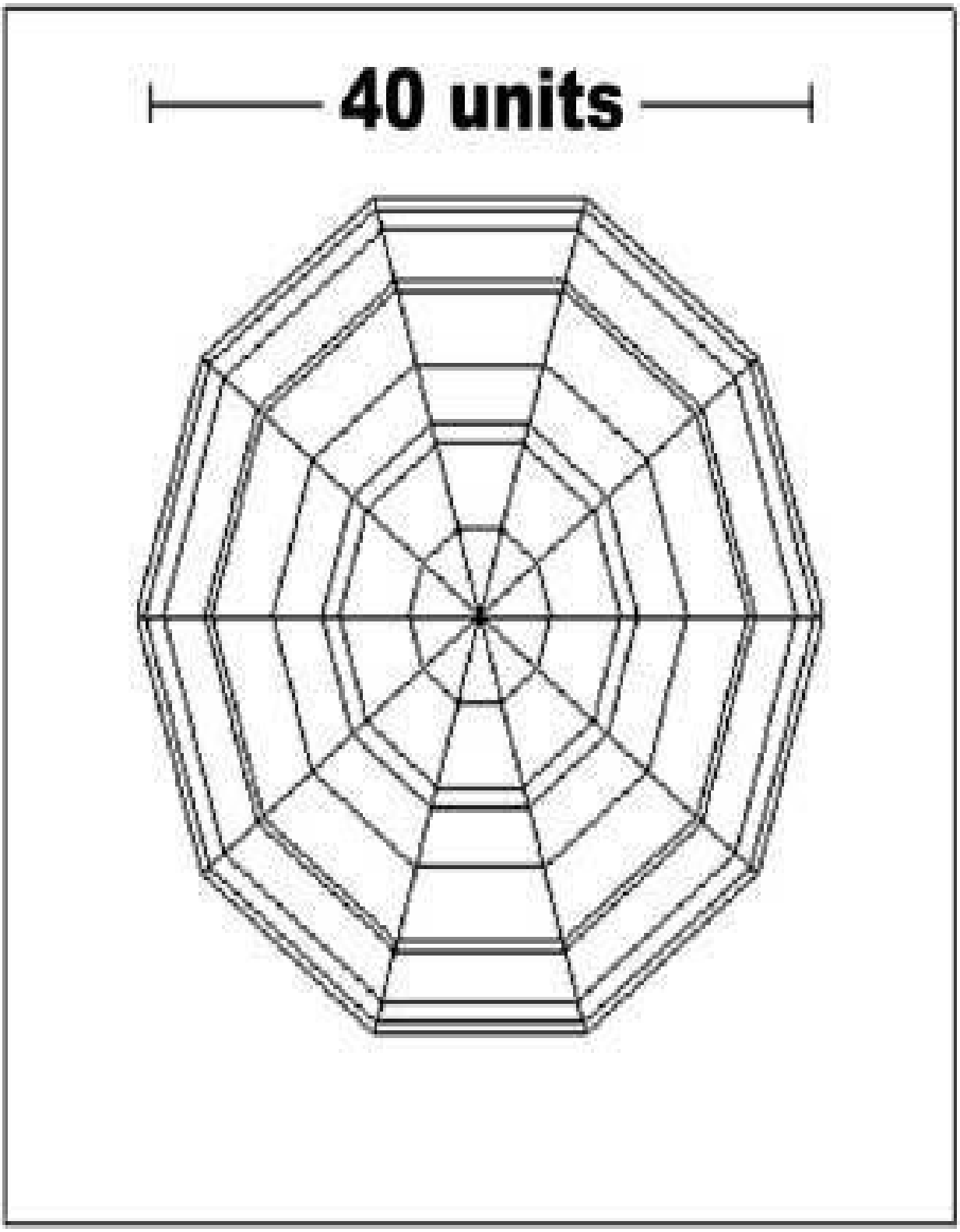

60 units
40 units

Figura 11-11

Ampliación de los bordes de la esfera.

Conceptos básicos

El método random() pertenece al objeto nativo Math. Devuelve un número aleatorio entre 0 y 1. Los límites pueden ser personalizados con la fórmula Math.random () * (Max - min) + min, donde min es el valor más pequeño que queremos conseguir y max el mayor. Por ejemplo, la fórmula Math.random () * (50 − 20) + 20 devuelve un número aleatorio entre 20 y 50.

La última operación del método draw() en el Código 11-17 representa la escena en el lienzo. Aunque esto es importante, no es lo último que tenemos que hacer. Todavía tenemos que mejorar el sistema de detección de colisiones. El coche puede estrellarse contra cualquiera de las cuatro paredes y las esferas verdes. Para detectar y responder a estas situaciones, hemos creado el método detect().

```
mijuego.detect = function(){
  var posx = mijuego.coche.malla.position.x;
  var posz = mijuego.coche.malla.position.z;

  if(posx < -980 || posx > 980 || posz < -980 || posz > 980){
    mijuego.coche.speed = -7;
  }

  if(posx > mijuego.targets.bordes[0] && posx < mijuego.targets.
 bordes[1] && posz > mijuego.targets.bordes[2] && posz < mijuego.
 targets.bordes[3]){
    mijuego.scene.remove(mijuego.targets.malla);
    mijuego.targets.malla = '';
  }
};
```

Código 11-18

Definición del método detect().

Al comparar las coordenadas actuales del coche con la posición de las paredes y los bordes de la zona que rodea a la esfera, se determinan las posibles colisiones. Si el vértice de la posición de del coche está fuera del área de juego (más allá de las paredes), el valor de la propiedad speed se establece en 7 negativo, haciendo que el coche rebote contra la pared. En el caso en que este vértice caiga dentro del área de la esfera, la malla se retira del espacio 3D por el método remove() y la propiedad targets.mesh se establece como null (nulo). Cuando esto sucede, la función draw() del Código 11-17 detecta que no hay ninguna esfera y dibuja una nueva.

La detección es la última característica que necesita nuestro juego. Pero se suele decir: “no se ha terminado hasta que se acaba”. Aún hay que construir un bucle que llame a los métodos una y otra vez, y añadir el detector para el evento load para empezar a ejecutar la aplicación.

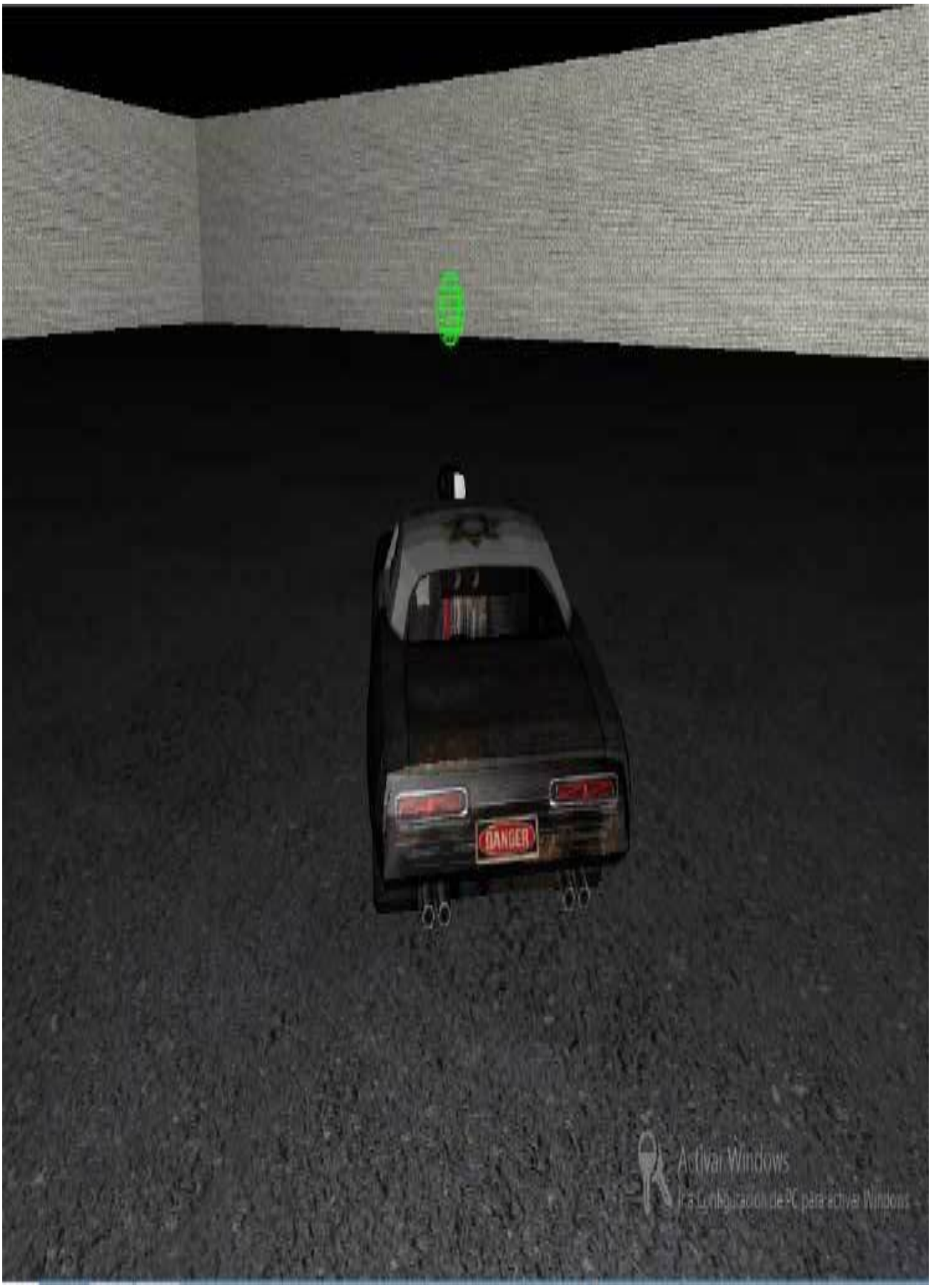

DANGER
Activar Windows
Ir a Configuración de PC para activar Windows.

Figura 11-12

Videojuego 3D para la Web. Modelo y texturas proporcionadas por TurboSquid Inc. (www.turbosquid.com).

```
mijuego.loop = function(){
   mijuego.control();
   mijuego.process();
   mijuego.detect();
   mijuego.draw();

   webkitRequestAnimationFrame(function(){ mijuego.loop() });
};
addEventListener('load', function(){ mijuego.iniciar() });
```

Código 11-19

Definición del método loop().

El código Javascript de este ejemplo está muy simplificado. No hay final del juego, ni puntos que celebrar, ni vidas que perder. La buena noticia es que ya sabe todo lo que necesita saber para agregar esas características, personalizar el juego y adecuarlo a su gusto.

Hágalo usted mismo

Todos los códigos que se presentan en esta parte del capítulo trabajan juntos para crear el videojuego, incluyendo el documento HTML del Código 11-10. Trate de aplicar la API Fullscreen a este ejemplo para mostrar su videojuego a pantalla completa.

Importante

Las mallas, texturas y modelos de los ejemplos de este capítulo no son libres de derechos y no incluyen los derechos para su libre distribución. Por favor, consulte a TurboSquid Inc. (www.turbosquid.com) o la Fundación Blender (www.blender.org) antes de utilizar este material para sus propios proyectos.

12 API Pointer Lock

12.1 Nuevo puntero del ratón

Las aplicaciones visuales, ya sean creadas con el elemento <canvas> o con WebGL, a veces exigen el uso de un método alternativo para expresar el movimiento del ratón. Hay muchas razones para esto: desde la necesidad de reemplazar la imagen del puntero del ratón, hasta la necesidad de modificar la posición de la cámara en una escena 3D sin mostrar una flecha en movimiento. Algunas aplicaciones ocultan el puntero del ratón para no afectar el diseño gráfico ni distraer al usuario de una imagen o un vídeo, cuando el uso del ratón no es necesario. Para adaptarse a estas exigencias, HTML5 introduce la API Pointer Lock.

12.1.1 Capturar el ratón

La API es un pequeño grupo de propiedades, métodos y eventos que ayudan a la aplicación a asumir el control del puntero del ratón. Proporciona dos métodos para bloquear y desbloquear el ratón:

requestPointerLock(): Este método bloquea el ratón y lo enlaza a un elemento específico.

exitPointerLock(): Este método desbloquea el ratón y hace que el puntero del ratón sea visible de nuevo.

Cuando se llama al método requestPointerLock(), el gráfico que representa el puntero (por lo general una pequeña flecha) está oculto y el código se encarga de proporcionar las pistas visuales para que el usuario pueda seguir interactuando con la aplicación a través del ratón.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>API Pointer Lock</title>
  <script>
    function iniciar(){
      var elemento = document.getElementById('aplicacion');
      elemento.addEventListener('click', bloquear);
    }
    function bloquear(e){
      var elemento = e.target;
      elemento.webkitRequestPointerLock();
    }
    addEventListener('load', iniciar);
  </script>
</head>
<body>
  <section id="aplicacion">

    Pulse aquí para bloquear el ratón.
  </section>
</body>
</html>
```

Código 12-1

Controlar el puntero del ratón.

A menos que el elemento que solicita el control del ratón se encuentre en modo de pantalla completa, el método requestPointerLock() devolverá un error si se llama sin intervención del usuario. Un movimiento del usuario, como podría ser un clic de ratón, debe preceder a la acción. Teniendo en cuenta esto, el Código 12-1 añade un detector para el evento click del elemento <section>. Cuando el usuario hace clic en este elemento, es llamada la función bloquear() y el método requestPointerLock() es utilizado para bloquear el ratón. En ese momento, el puntero del ratón desaparece de la pantalla y no se muestra de nuevo hasta que el usuario cambia a otra ventana o cancela el modo pulsando la tecla Escape en el teclado. Cuando el ratón está bloqueado por un elemento, el resto de los elementos no dispara ningún evento de ratón hasta que el mismo es desbloqueado mediante la aplicación o el modo es cancelado por el usuario. La API incluye un evento para informar cuando un elemento ha bloqueado el ratón y otro para corregir errores:

pointerlockchange: Este evento se desencadena cuando un elemento bloquea o desbloquea el ratón.

pointerlockerror: Este evento se activa cuando el intento de bloquear el ratón falla. Este evento no devuelve ningún dato.

Importante

Los ejemplos de este capítulo se aplican a las propiedades, métodos y eventos específicos de Google Chrome. Pointer Lock es una API experimental y está siendo implementada en los navegadores mientras preparamos este libro. Los navegadores en los que esta API está disponible son Google Chrome y Mozilla Firefox. Para Mozilla Firefox, las propiedades, métodos y eventos son: mozRequestPointerLock(), mozExitPointerLock(), mozpointerlockchange, mozpointerlockerror y mozFullscreenElement.

12.1.2 pointerLockElement

Esta API incluye la propiedad pointerLockElement para informar de qué elemento ha bloqueado al ratón. Si no hay ningún elemento que esté controlando al ratón, se devuelve el valor null.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>API Pointer Lock</title>
  <script>
    var lienzo;
    function iniciar(){
      var elemento = document.getElementById('lienzo');
      lienzo = elemento.getContext('2d');

      elemento.addEventListener('click', bloquear);
      elemento.addEventListener('mousemove', dibujar);
    }
    function dibujar(e){
      lienzo.clearRect(0, 0, 500, 400);
      var posx = e.clientX;
      var posy = e.clientY;
      lienzo.beginPath();
      lienzo.moveTo(posx, posy - 20);
      lienzo.lineTo(posx, posy + 20);
      lienzo.moveTo(posx - 20, posy);
      lienzo.lineTo(posx + 20, posy);
      lienzo.moveTo(posx + 20, posy);
      lienzo.arc(posx, posy, 20, 0, Math.PI * 2);
      lienzo.stroke();
    }
    function bloquear(e){
      var elemento = e.target;
      if(!document.webkitPointerLockElement){
        elemento.webkitRequestPointerLock();
      }else{
        document.webkitExitPointerLock();
      }
    }
    addEventListener('load', iniciar);
  </script>
</head>
<body>
  <section id="aplicacion">
    <canvas id="lienzo" width="500" height="400"></canvas>
  </section>
</body>
</html>
```

Código 12-2

Bloqueo y desbloqueo del ratón.

En este ejemplo hemos creado una pequeña aplicación para mostrar un enfoque más realista de la utilización de esta API. Usamos la propiedad pointerLockElement para determinar si el lienzo controla el ratón o no. Cada vez que el usuario hace clic en el elemento <canvas>, se comprueba esta condición y, en consequencia, se bloquea o se desbloquea el ratón sando requestPointerLock() o exitPointerLock() según corresponda.

Cuando el ratón está bloqueado, el navegador asigna el control del ratón al elemento que hace la petición. Todos los eventos del ratón, como mousemove, click o mousewheel, son activados solo para este elemento, pero los acontecimientos relacionados con el puntero del ratón, como tales como mouseover o mouseout, ya no son activados. Como resultado, para interactuar con el ratón y detectar sus movimientos, hay que detectar el evento mousemove. El Código 12-2 añade un detector para el evento mousemove, para dibujar un punto de mira en el lienzo en la posición devuelta por las propiedades clientX y clientY. Estas propiedades de eventos devuelven las coordenadas del puntero del ratón, en píxeles, con respecto a la esquina superior izquierda de la ventana. Aunque esto es útil en algunas circunstancias, se limita la acción y el control del puntero del ratón dentro de los límites del elemento o de la ventana. La API ofrece dos nuevas propiedades para el evento mousemove que nos permitirán calcular el movimiento del ratón, no importa lo lejos que se desplace.

12.1.3 movementX y movementY

Las propiedades movementX y movementY difieren de sus hermanas, clientX y clientY, en el tipo de valor devuelto. Mientras clientX y clientY devuelven la posición del puntero del ratón según una cuadrícula de píxeles que comienza a partir de las coordenadas 0, 0 en la esquina superior izquierda de la ventana, las propiedades movementX y movementY proporcionan valores que representan solo el cambio en la posición (la diferencia entre la posición anterior y la actual). Es decir, la atención se centra en el desplazamiento y no en una posición específica. Este enfoque tiene la ventaja de que permite procesar constantemente el movimiento del ratón de la misma manera, sin importar donde se encuentra éste o cuánto se ha movido en una dirección u otra.

En el ejemplo del Código 12-2, el punto de mira se mueve con el puntero del ratón, pero tan pronto como el ratón se bloquea, el punto de mira se congela en la pantalla. Esto es porque cuando el ratón está bloqueado, solo se actualizan los valores de movementX y movementY (el resto de las propiedades de evento mantienen los últimos valores almacenados antes de la activación del modo).

Para poder trabajar con el ratón cuando está bloqueado y aprovechar el nivel de control proporcionado por estas propiedades, hay que implementarlos de la siguiente manera:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>API Pointer Lock</title>
  <script>
    var lienzo, posx, posy;
    function iniciar(){
      var elemento = document.getElementById('lienzo')
      lienzo = elemento.getContext('2d');

      elemento.addEventListener('click', bloquear);
      iniciarmensaje();
    }
    function dibujar(e){
      lienzo.clearRect(0, 0, 500, 400);

      var test1, test2;
      test1 = posx + e.webkitMovementX;
      test2 = posy + e.webkitMovementY;
      if(test1 > 0 && test1 < 500){
        posx = test1;
      }
      if(test2 > 0 && test2 < 400){
        posy = test2;
      }
      lienzo.beginPath();
      lienzo.moveTo(posx, posy - 20);
      lienzo.lineTo(posx, posy + 20);
      lienzo.moveTo(posx - 20, posy);
      lienzo.lineTo(posx + 20, posy);
      lienzo.moveTo(posx + 20, posy);
      lienzo.arc(posx, posy, 20, 0, Math.PI * 2);
      lienzo.stroke();
    }
    function bloquear(e){
      var elemento = e.target;
      if(!document.webkitPointerLockElement){
        elemento.webkitRequestPointerLock();
        posx = e.clientX;
        posy = e.clientY;
        elemento.addEventListener('mousemove', dibujar);
      }else{
        document.webkitExitPointerLock();
        elemento.removeEventListener('mousemove', dibujar);
        iniciarmensaje();
      }
    }
    function iniciarmensaje(){
      lienzo.clearRect(0, 0, 500, 400);
      lienzo.font = "bold 36px verdana, sans-serif";
      lienzo.fillText('Click to Start', 120, 180);
    }
    addEventListener('load', iniciar);
  </script>
</head>
<body>
  <section id="aplicacion">
    <canvas id="lienzo" width="500" height="400"></canvas>
  </section>
</body>
</html>
```

Código 12-3

Cálculo de la posición del puntero del ratón con movementX y movementsY.

Hemos hecho algunos cambios en el guión para aumentar el nivel de control y demostrar cómo sacar provecho de esta API en ciertas fases de la aplicación. La función iniciar() ajusta el lienzo, añade un detector para el evento click para bloquear el ratón y llama a la función iniciarmensaje() para mostrar un mensaje de bienvenida en la pantalla, pero no se añade un detector para el evento mousemove. Este detector se añade en la función bloquear() después de que el ratón es bloqueado y se elimina en la misma función cuando el ratón es desbloqueado. Después de este procedimiento, cuando ya se ha ejecutado la primera etapa de la aplicación (en la que pide al usuario que haga clic en la pantalla), el elemento <canvas> obtiene el control del ratón. Si el usuario vuelve a esta primera etapa, el ratón se desbloquea de nuevo y el detector para el acontecimiento mousemove se elimina gracias al método removeEventListener().

Los valores devueltos por las propiedades movementX y movementY reflejan el cambio en la posición del ratón. Antes de que el ratón sea bloqueado, podemos capturar las coordenadas con las propiedades clientX y clientY para establecer la posición inicial del punto de mira. Por lo general, esto no es necesario, pero en una aplicación como ésta, donde el ratón es constantemente bloqueado y desbloqueado, ayuda a producir una mejor transición de un estado a otro.

Esta posición inicial se almacena en las variables posx y posy y solo es modificada por el valor del desplazamiento mientras no se superen los límites establecidos por el elemento <canvas> (consulte la declaración if en la función dibujar()). Este control es necesario para evitar almacenar valores que la aplicación no puede procesar, salvo que solo se quiera comprobar la dirección

del ratón.

Importante

Para el momento en el que estamos escribiendo este manual, las propiedades movementX y movementY utilizan prefijos en Google Chrome y Mozilla Firefox: webkitMovementX, webkit-MovementY, mozMovementX, mozMovementY.

Hágalo usted mismo

Cree un archivo HTML y pruebe los ejemplos de este capítulo que prefiera.

13 API Drag and Drop

13.1 Arrastrar y soltar en la web

Arrastrar un elemento desde un lugar y luego soltarlo en otro es algo que hacemos todo el tiempo en aplicaciones de escritorio, pero ni siquiera imaginamos hacerlo en la web. Esto no es debido a que las aplicaciones web sean diferentes sino porque los desarrolladores nunca contaron con una tecnología estándar para ofrecer esta herramienta. Ahora, gracias a la API Drag and Drop, introducida por la especificación HTML5, finalmente tenemos la oportunidad de crear un software para la web que se comporte exactamente igual que las aplicaciones de escritorio que usamos desde siempre.

13.1.1 Eventos

Uno de los aspectos más importantes de esta API es un conjunto de siete nuevos eventos introducidos para informar diferentes situaciones involucradas en el proceso. Algunos de estos eventos son disparados por la fuente (el elemento arrastrado) y otros son disparados por el destino (el elemento en el cual el será soltada la fuente). Por ejemplo, cuando el usuario realiza una operación de arrastrar y soltar, el elemento fuente dispara estos tres eventos:

dragstart: Este evento es disparado en el momento en el que el arrastre comienza. Los datos asociados con el elemento de origen son definidos en este momento en el sistema.

drag: Es similar al evento mousemove, excepto que será disparado durante una operación de arrastre por el elemento de origen.

dragend: Cuando la operación de arrastrar y soltar finaliza (sea la operación exitosa o no), este evento es disparado por el elemento de origen.

Y estos son los eventos disparados por el elemento de destino (donde será soltado el elemento de origen) durante la operación:

dragenter: Es disparado cuando el puntero del ratón entra dentro del área ocupada por los posibles elementos destino durante una operación de arrastre.

dragover: Este evento es similar al evento mousemove, excepto que es disparado durante una operación de arrastre por posibles elementos de destino.

drop: Es disparado cuando el elemento de origen es soltado.

dragleave: Es disparado cuando el ratón sale de un elemento durante una operación de arrastre. Este evento es generalmente usado junto con el evento dragenter para devolver una respuesta y mostrar una ayuda visual al usuario, que le permita identificar el elemento de destino.

Antes de trabajar con esta nueva herramienta, existe un aspecto importante que debemos considerar. Los navegadores realizan acciones por defecto durante una operación de arrastrar y soltar. Para obtener el resultado que queremos, hay que evitar este comportamiento predeterminado del navegador. Esto es necesario para algunos eventos, como dragenter, dragover y drop, aunque se haya especificado una acción personalizada. Veamos cómo debemos proceder usando un ejemplo:

```
<!DOCTYPE html>
<html lang="es">
<head>
 <title>Drag and Drop</title>
 <link rel="stylesheet" href="dragdrop.css">
 <script src="dragdrop.js"></script>
</head>
<body>
 <section id="cajasoltar">
  Arrastre y suelte la imagen aquí
 </section>
 <section id="cajaimagenes">
  <img id="imagen" src="http://www.minkbooks.com/content/
monster1.gif">
 </section>
</body>
</html>
```

Código 13-1

Documento HTML para probar la API Drag and Drop.

El documento HTML del Código 13-1 incluye un elemento <section> identificado como cajasoltar y una imagen. El elemento <section> es usado como elemento de destino y la imagen como elemento fuente. También incluimos dos archivos para estilos CSS y el código Javascript que se hará cargo de la operación.

```
#cajasoltar{
 float: left;
 width: 500px;
 height: 300px;
 margin: 10px;
 border: 1px solid #999999;
}
#cajaimagenes{
 float: left;
 width: 320px;
 margin: 10px;
 border: 1px solid #999999;
}
#cajaimagenes > img{
float: left;
padding: 5px;
}
```

Código 13-2

Estilos para la plantilla (dragdrop.css).

Las reglas del Código 13-2 simplemente otorgan estilos a las cajas que nos permitirán identificar el elemento a arrastrar y la caja de destino. A continuación presentamos el código Javascript que controlará toda la operación.

```
var origen1, destino;
function iniciar(){
 origen1=document.getElementById('imagen');
 origen1.addEventListener('dragstart', arrastrado);

 destino = document.getElementById('cajasoltar');
 destino.addEventListener('dragenter', function(e){
 e.preventDefault(); });
 destino.addEventListener('dragover', function(e){e.preventDefault();
 });
 destino.addEventListener('drop', soltado);
}
function arrastrado(e){
 var codigo = '<img src ="' + origen1.getAttribute('src') + '">';
 e.dataTransfer.setData('Text', codigo);
}
function soltado(e){
 e.preventDefault();
 destino.innerHTML = e.dataTransfer.getData('Text');
}
addEventListener('load', iniciar);
```

Código 13-3.

Código elemental para una operación de arrastrar y soltar.

En el Código 13-3, presentamos tres funciones: la función iniciar() agrega detectores de eventos, y las funciones arrastrado() y soltado() generan y reciben la información que es transmitida por este proceso.

Para que una operación de arrastrar y soltar se realice normalmente, debemos preparar la información que será compartida entre el elemento de origen y el elemento de destino. Para lograr esto, es agregado un detector para el evento dragstart. La función arrastrado(), que es la que actúa como detector del evento, prepara la información que va a ser compartida usando el método setData(), exclusivo de esta API. La operación de soltar no es normalmente permitida en muchos elementos por defecto. Por este motivo, para hacer que esta operación esté disponible para nuestro elemento de destino, debemos evitar el comportamiento por defecto del navegador. Esto se hace agregando un detector para los eventos dragenter y dragover y ejecutando el método preventDefault() con una función anónima. Finalmente, agregamos un detector del evento drop para llamar a la función soltado() que recibirá y procesará los datos enviados por el elemento de origen.

Conceptos básicos

Para responder a los eventos dragenter y dragover, usamos una función anónima que aplica el método preventDefault() y cancela el comportamiento por defecto del navegador. La variable e es enviada como un atributo para hacer referencia al

evento en cuestión dentro de la función anónima.

Cuando el usuario comienza a arrastrar el elemento de origen, el evento dragstart es disparado y es llamada la función arrastrado(). En esta función obtenemos el valor del atributo src del elemento arrastrado y declaramos los datos que serán transferidos usando el método setData() del objeto dataTransfer. Desde el otro lado, cuando un elemento es soltado dentro del elemento de destino, el evento drop es disparado y es llamada la función soltado(). Esta función modifica el contenido del elemento de destino con la información obtenida por el método getData(). Los navegadores también realizan acciones por defecto cuando este evento se activa (por ejemplo, al abrir un enlace o actualizar la ventana para mostrar la imagen que fue soltada) por lo que debemos evitar este comportamiento usando el método preventDefault(), como ya hicimos para otros eventos anteriormente.

Hágalo usted mismo

Cree un archivo HTML con la plantilla del Código 13-1, un archivo CSS llamado dragdrop.css con los estilos del Código 13-2, y un archivo Javascript llamado dragdrop.js con el código del Código13-3. Para probar el ejemplo, abra el archivo HTML en su navegador y arrastre la imagen hacia el cuadro de la izquierda.

13.1.2 DataTransfer

Éste es el objeto que contendrá la información en una operación arrastrar y soltar. El objeto DataTransfer tiene varios métodos y propiedades asociados. Ya utilizamos los métodos setData() y getData() en nuestro ejemplo del Código 13-3. Junto con clearData(), estos son los métodos a cargo de la información que es transferida:

setData(tipo, dato): Este método es usado para declarar los datos a ser enviados y su tipo. El método puede recibir tipos de datos regulares (como text/plain, text/html o text/uri-list), tipos de datos especiales (como URL o Text) o incluso tipos de datos personalizados. para cada tipo de datos que queremos enviar en la misma operación, hay que llamar a un método setData().

getData(tipo): Este método retorna los datos del tipo especificado enviados por el elemento de origen.

clearData(): Este método elimina los datos del tipo especificado.

Importante

Podríamos haber usado un tipo de datos más apropiado en nuestro ejemplo, como text/html o incluso un tipo personalizado, pero varios navegadores solo admiten un número limitado de tipos en este momento, por lo que el tipo Text

hace a nuestra pequeña aplicación más compatible y la deja lista para ser ejecutada.

En la función arrastrado() del Código 13-3, creamos un pequeño código HTML que incluye el valor del atributo src del elemento que disparó el evento dragstart (es decir, el elemento arrastrado), grabamos este código en la variable codigo y luego enviamos esta variable como el dato a ser transferido usando el método setData(). Debido a que estamos enviando texto, declaramos el tipo de dato como Text.

Cuando recuperamos los datos en la función soltado() usando el método getData(), debemos especificar el tipo de datos que será leído. Esto es debido a que pueden ser enviados por el mismo elemento y en el mismo elemento diferentes clases de datos. Por ejemplo, una imagen podría enviar la imagen misma, la URL y un texto que la describe. Toda esta información puede ser enviada usando varios métodos setData() con diferentes tipos de valores y luego puede ser recuperada por método getData() especificando los mismo tipos.

Importante

Para obtener mayor información acerca de tipos de datos válidos para la operación de arrastrar y soltar, visite nuestro sitio web y siga los enlaces correspondientes a este capítulo.

El objeto dataTransfer tiene algunos métodos y propiedades más que a veces podrían resultar útiles para nuestras aplicaciones:

setDragImage(elemento, x, y): Este método es usado para personalizar la imagen en miniatura junto al puntero del ratón que muestran algunas aplicaciones durante el arrastre y seleccionar la posición en relación al puntero del ratón. Esta posición está determinada por los atributos x e y.

types: Esta propiedad retorna una matriz que contiene los tipos de datos que fueron declarados en el evento dragstart (por el código o por el navegador). Podemos grabar esta matriz en una variable (por ejemplo, lista=dataTransfer.types) y luego leerlo con un bucle for.

files: Esta propiedad retorna una matriz que contiene información acerca de los archivos que están siendo arrastrados.

dropEffect: Esta propiedad retorna el tipo de operación actualmente seleccionada. Los posibles valores son none, copy, link y move.

effectAllowed: Esta propiedad devuelve o establece los tipos de operaciones que están permitidas. Los posibles valores son: none, copy, copyLink, copy-Move, link, linkMove, move, all y uninitialized.

Aplicaremos algunos de estos métodos y propiedades en los siguientes ejemplos.

13.1.3 dragenter, dragleave y dragend

De momento solo hemos usado el evento dragenter para cancelar el comportamiento por defecto del navegador. Tampoco hemos sacado provecho de los eventos dragleave y dragend. Estos son eventos importantes que nos permitirán ayudar al usuario cuando esté arrastrando objetos por la pantalla.

```
var origen1, soltar
function iniciar(){
  origen1 = document.getElementById('imagen');
  origen1.addEventListener('dragstart', arrastrado);
  origen1.addEventListener('dragend', finalizado);

  soltar = document.getElementById('cajasoltar');
  soltar.addEventListener('dragenter', entrando);
  soltar.addEventListener('dragleave', saliendo);
  soltar.addEventListener('dragover', function(e){
 e.preventDefault(); });
  soltar.addEventListener('drop', soltado);
}

function entrando(e){
  e.preventDefault();
  soltar.style.background = 'rgba(0, 150, 0, .2)';
}
function saliendo(e){
  e.preventDefault();
  soltar.style.background = '#FFFFFF';
}
function finalizado(e){
  elemento = e.target;
  elemento.style.visibility = 'hidden';
}

function arrastrado(e){
  var codigo='<img src="'+origen1.getAttribute('src')+'">';
  e.dataTransfer.setData('Text', codigo);
}
function soltado(e){
  e.preventDefault();
  soltar.style.background='#FFFFFF';
  soltar.innerHTML = e.dataTransfer.getData('Text');
}
addEventListener('load', iniciar);
```

Código 13-4

Controlar todo el proceso de arrastrar y soltar.

El Código 13-4 reemplazan al Código 13-3. En este nuevo ejemplo, agregamos dos funciones para el elemento de destino y una para el elemento de origen. Las funciones entrando() y saliendo() cambian el color de fondo de la caja de destino cada vez que el puntero del ratón arrastra un objeto y entra o sale del área ocupada por este elemento, pues estas acciones disparan los eventos dragenter y dragleave. Además, la función finalizado() es llamada por el detector del evento dragend cuando el objeto arrastrado es soltado. Note que ni este evento ni la función controlan si el proceso fue exitoso o no. Esto hay que controlarlo a través del código. Gracias estas funciones, cada vez que el ratón arrastra un objeto y entra en el área de la caja de destino, ésta adquiere color verde, y cuando el objeto es soltado, la imagen original se oculta. Estos cambios visibles no afectan el proceso de arrastrar y soltar, pero sí ofrecen al usuario la operación una información muy clara.

Hágalo usted mismo

Copie el texto del Código 13-4 dentro del archivo Javascript, abra el documento HTML del Código 13-1 en su navegador, y arrastre la imagen que aparece en la pantalla dentro de la caja vacía.

Para evitar acciones predeterminadas de la navegador, tenemos que usar el método preventDefault() en cada función, aunque hayamos declarado acciones

personalizadas.

13.1.4 Seleccionar una fuente válida

No existe ningún método específico para detectar si el elemento de origen es válido o no. No podemos confiar en la información retornada por el método getData() porque aunque podemos recuperar solo los datos del tipo especificado por el atributo, otras fuentes podrían usar el mismo tipo y proveer datos que no esperábamos. Hay una propiedad del objeto DataTransfer llamada types que retorna una matriz con la lista de tipos establecidos durante el evento dragstart, pero también es inútil para propósitos de validación.

Por esta razón, las técnicas para seleccionar y validar datos transferidos en una operación de arrastrar y soltar son variadas, y pueden ser tan simples o tan complejas como lo requiera nuestra aplicación.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Drag and Drop</title>
  <link rel="stylesheet" href="dragdrop.css">
  <script src="dragdrop.js"></script>
</head>

<body>
  <section id="cajasoltar">
   Arrastre y suelte las imágenes aquí
  </section>
  <section id="cajaimagenes">
    <img id="imagen1"src="http://www.minkbooks.com/content/
                          monster1.gif">
    <img id="imagen2"src="http://www.minkbooks.com/content/
                          monster2.gif">
    <img id="imagen3"src="http://www.minkbooks.com/content/
                          monster3.gif">
    <img id="imagen4"src="http://www.minkbooks.com/content/
                          monster4.gif">
  </section>
</body>
</html>
```

Código 13-5

Nueva plantilla con varias fuentes.

Usando la nueva plantilla HTML del Código 13-5, vamos a filtrar los elementos de origen controlando el atributo id de la imagen. El siguiente código Javascript indicará cuál imagen puede ser soltada y cuál no:

```
var soltar;
function iniciar(){
  var imagenes = document.querySelectorAll('#cajaimagenes > img');
  for(var i = 0; i<imagenes.length; i++){
    imagenes[i].addEventListener('dragstart', arrastrado);
  }
  soltar = document.getElementById('cajasoltar');
  soltar.addEventListener('dragenter', function(e){
 e.preventDefault(); });
  soltar.addEventListener('dragover', function(e){
e.preventDefault(); });
soltar.addEventListener('drop', soltado);
}
function arrastrado(e){
  elemento = e.target;
  e.dataTransfer.setData('Text', elemento.getAttribute('id'));
}
function soltado(e){
  e.preventDefault();
  var id = e.dataTransfer.getData('Text');
  if(id!="imagen4"){
    var src = document.getElementById(id).src;
    soltar.innerHTML = '<img src="'+src+'">';
  }else{
    soltar.innerHTML = 'la imagen no es admitida';
}
} addEventListener('load', iniciar);
```

Código 13-6

Enviar el valor del atributo id.

No han cambiado muchas cosas en el Código 13-6 respecto a los códigos anteriores. En este código usamos el método querySelectorAll() para agregar un detector para el evento dragstart a cada una de las imágenes que se encuentran dentro del elemento cajaimagenes, enviamos el valor del atributo id con setData() cada vez que una imagen es arrastrada y confirmamos el valor su id en la función soltado() para evitar que el usuario arrastre y suelte la imagen cuyo valor del atributo id es imagen4. El mensaje la imagen no es admitida es mostrado dentro de la caja de destino cuando el usuario intenta arrastrar y soltar esta imagen en particular.

Éste es, por supuesto, un filtro extremadamente sencillo. Puede usar el método querySelectorAll() en la función soltado() para controlar que la imagen recibida sea una de las que se encuentran dentro del elemento cajaimagenes, por ejemplo, o usar propiedades del objeto DataTransfer (como types o files), pero es siempre un proceso personalizado. En otras palabras, deberá hacerse cargo usted mismo de realizar este control.

13.1.5 setDragImage()

Cambiar la imagen en miniatura que es mostrada junto al puntero del ratón en una operación de arrastrar y soltar puede parecer inútil, pero en ocasiones nos evitará dolores de cabeza. El método setDragImage() no solo nos permite cambiar la imagen sino también recibe dos atributos, x e y, para especificar la posición de esta imagen respecto al puntero del ratón. Algunos navegadores generan una imagen en miniatura por defecto a partir del objeto original que es arrastrado, pero su posición respecto al puntero del ratón es determinada por la posición del puntero cuando el proceso comienza. El método setDragImage() nos permite declarar una posición específica que será la misma para cada operación de arrastrar y soltar.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Drag and Drop</title>
  <link rel="stylesheet" href="dragdrop.css">
  <script src="dragdrop.js"></script>
</head>
<body>
  <section id="cajasoltar">
  <canvas id="lienzo" width="500" height="300"></canvas>
  </section>
  <section id="cajaimagenes">
<img id="imagen1"
   src="http://www.minkbooks.com/content/monster1.gif">
   <img id="imagen2"src="http://www.minkbooks.com/content
                        /monster2.gif">
<img id="imagen3"src="http://www.minkbooks.com/content
                        /monster3.gif">

  <img id="imagen4"
  src="http://www.minkbooks.com/content/monster4.gif">
  </section>
  </body>
  </html>
```

Código 13-7

Uso del elemento <canvas> como elemento de destino.

Con el nuevo documento HTML del Código 13-7 vamos a demostrar la importancia del método setDragImage() usando un elemento <canvas> como destino.

```
var soltar, lienzo
function iniciar(){
  var imagenes = document.querySelectorAll('#cajaimagenes > img');
  for(var i = 0; i < imagenes.length; i++){
    imagenes[i].addEventListener('dragstart', arrastrado);
    imagenes[i].addEventListener('dragend', finalizado);
  }
  soltar = document.getElementById('lienzo');
  sienzo = soltar.getContext('2d');

  soltar.addEventListener('dragenter', function(e){
  e.preventDefault(); });
  soltar.addEventListener('dragover', function(e){
  e.preventDefault(); });
  soltar.addEventListener('drop', soltado);
}
function finalizado(e){
  elemento = e.target;
  elemento.style.visibility='hidden';
}
function arrastrado(e){
  elemento = e.target;
  e.dataTransfer.setData('Text', elemento.getAttribute('id'));
  e.dataTransfer.setDragImage(elemento, 0, 0);
}
function soltado(e){
  e.preventDefault();
  var id = e.dataTransfer.getData('Text');
  var elemento = document.getElementById(id);
  var posx = e.pageX - soltar.offsetLeft;
  var posy = e.pageY - soltar.offsetTop;
  lienzo.drawImage(elemento, posx, posy);
}
addEventListener('load', iniciar);
```

Código 13-8

Una pequeña aplicación para arrastrar y soltar.

Con este ejemplo probablemente nos estemos acercando a lo que sería una aplicación de la vida real. El Código 13-8 controla tres diferentes aspectos del proceso. Cuando la imagen es arrastrada, se llama a la función arrastrado() y se genera en su interior una imagen miniatura con el método setDragImage(). El código también crea el contexto para trabajar con el lienzo y dibuja la imagen soltada usando el método drawImage() y la referencia del origen. Al final de todo el proceso se oculta la imagen original usando la función finalizado().

Para crear la imagen miniatura personalizada usamos el mismo elemento que está siendo arrastrado, pero declaramos la posición respecto al puntero del ratón como 0,0. Gracias a esto ahora sabremos siempre la ubicación exacta de la imagen miniatura. Este dato lo utilizamos dentro de la función soltado(). Usando la misma técnica introducida en capítulos anteriores, calculamos en qué parte del lienzo fue soltado el objeto y dibujamos la imagen en ese lugar preciso. Si prueba este ejemplo en navegadores que ya aceptan el método setDragImage(), como puede ser Mozilla Firefox, verá que la imagen es dibujada en el lienzo exactamente en la posición en la que es soltada la imagen miniatura, de manera que el usuario pueda escoger fácilmente el lugar adecuado para soltar la imagen.

Importante

El Código 13-8 usa el evento dragend para ocultar la imagen original cuando termina la operación. Este evento es disparado por el elemento de origen cuando

una operación de arrastre finaliza, incluso si ésta no es exitosa. En nuestro ejemplo la imagen es ocultada tanto si el arrastre es exitoso como si no lo es. Usted deberá crear los controles adecuados para proceder solo en caso de éxito.

13.1.6 Archivos

Posiblemente la característica más interesante de la API Drag and Drop es su habilidad de trabajar con archivos. La API no está disponible solo dentro del documento, sino que también está integrada con el sistema, permitiendo a los usuarios arrastrar elementos desde el navegador hacia otras aplicaciones y viceversa. Y normalmente los elementos más requeridos desde aplicaciones externas son los archivos. Como vimos anteriormente, existe una propiedad específica para este objetivo en el objeto DataTransfer, que retornará una matriz que contiene la lista de archivos arrastrados. Podemos usar esta información para construir complejos códigos para ayudar a los usuarios a trabajar con archivos o a subirlos a un servidor.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Drag and Drop</title>
  <link rel="stylesheet" href = "dragdrop.css">
  <script src="dragdrop.js"></script>
</head>
<body>
  <section id="cajasoltar">
Arrastre y suelte archivos en este espacio
  </section>
</body>
</html>
```

Código 13-9

Plantilla simple para arrastrar archivos.

El documento HTML del Código 13-9 genera simplemente una caja para soltar los archivos arrastrados. Los archivos serán arrastrados desde una aplicación externa (por ejemplo, el Explorador de Archivos de Windows). Los datos provenientes de los archivos serán procesados por el siguiente código:

```
var soltar;
function iniciar(){
  soltar = document.getElementById('cajasoltar');
  soltar.addEventListener('dragenter', function(e){
  e.preventDefault(); });
  soltar.addEventListener('dragover', function(e){
  e.preventDefault(); });
  soltar.addEventListener('drop', soltado);
}
function soltado(e){
  e.preventDefault();
  var archivos = e.dataTransfer.files;
  var lista='';
  for(var f = 0; f < archivos.length; f++){
  lista += 'Archivo: '+archivos[f].name+' '+archivos[f].size+'<br>';
  }
  soltar.innerHTML=lista;
}
addEventListener('load', iniciar);
```

Código 13-10

Procesar los datos en la propiedad files.

La información devuelta por la propiedad files del objeto dataTransfer puede ser grabada en una variable y luego leída por un bucle for. El Código 13-10 solo muestra el nombre y el tamaño de cada archivo en el elemento de destino. Para aprovechar esta información en la construcción de aplicaciones más complejas, necesitaremos implementar la API Drag and Drop junto a otras API, como veremos más adelante en este libro.

Hágalo usted mismo

Cree nuevos archivos con los Códigos 13-9 y 13-10, y abra la plantilla en su navegador. Arrastre archivos desde el Explorador de archivos o cualquier otra aplicación similar hasta la caja de destino. Luego de esta acción debería ver en pantalla una lista con el nombre y tamaño de cada archivo arrastrado.

14 API Web Storage

14.1 Dos sistemas de almacenamiento

La Web fue primero pensada como una forma de mostrar información estática. El procesamiento de información comenzó luego, primero con aplicaciones del lado del servidor y más tarde, de forma bastante ineficiente, a través de pequeños códigos y complementos (plug-ins) ejecutados en el ordenador del usuario. Sin embargo, la esencia de la Web siguió siendo básicamente la misma: la información era preparada en el servidor y luego mostrada a los usuarios. El trabajo duro se desarrollaba del lado del servidor porque el sistema no aprovechaba los poderosos recursos disponibles en los ordenadores de los usuarios.

HTML5 equilibra esta situación. Justificada por las particulares características de los dispositivos móviles, el surgimiento de los sistemas de computación en la nube y la necesidad de estandarizar tecnologías e innovaciones introducidas por plug-ins a través de los últimos años, HTML5 incluye herramientas que hacen posible ejecutar aplicaciones completamente funcionales en el ordenador del usuario.

Una de las características más necesitadas en cualquier aplicación es la posibilidad de almacenar datos para disponer de ellos cuando sean necesarios y hasta hace poco no existía un mecanismo efectivo para este fin. Las llamadas cookies eran usadas para almacenar pequeñas cantidades de información del lado del cliente, pero debido a su naturaleza estaban limitadas a breves textos, lo que las hacía útiles solo en determinadas circunstancias.

La API Web Storage es básicamente una mejora de las cookies. Esta API permite almacenar datos en el disco duro del usuario y utilizarlos luego del mismo modo que lo haría una aplicación de escritorio. El proceso de

almacenamiento que proporciona esta API puede ser utilizado en dos situaciones particulares: cuando la información tiene que estar disponible solo durante una sesión, y cuando tiene que conservarse durante el tiempo que el usuario indique. Para hacer estos métodos más claros y comprensibles para los desarrolladores, la API ha sido dividida en dos partes llamadas sessionStorage y localStorage:

sessionStorage: Éste es un mecanismo de almacenamiento que permite que los datos estén disponibles solo mientras dure la sesión de una página. De hecho, a diferencia de las sesiones reales, la información almacenada a través de este mecanismo es accesible desde una única ventana o pestaña y es conservada hasta que la se cierra la ventana. La especificación aún habla de "sesiones" ya que la información es conservada incluso cuando la ventana es actualizada o es cargada una nueva página desde el mismo sitio web.

localStorage: Este mecanismo trabaja de forma similar a un sistema de almacenamiento para aplicaciones de escritorio. Los datos son guardados de forma permanente y se encuentran siempre disponibles para la aplicación que los creó en un principio.

Ambos mecanismos trabajan a través de la misma interfaz y comparten los mismos métodos y propiedades. Y ambos dependen del origen, lo que quiere decir que la información está disponible solo a través del sitio web que los creó. Cada sitio web debe tener asignado su propio espacio de almacenamiento, que puede durar hasta que la ventana sea cerrada o puede ser permanente, de acuerdo al mecanismo utilizado. La API diferencia claramente los datos temporales de permanentes y facilita la construcción de pequeñas aplicaciones que necesiten conservar pequeños textos como referencia temporal (por ejemplo, carros de compra), o aplicaciones más grandes y complejas que necesitan almacenar documentos completos por todo el tiempo que sea necesario.

Importante

Muchos navegadores solo trabajan de forma adecuada con esta API cuando la fuente es un servidor real. Para probar los siguientes códigos, le recomendamos que primero suba los archivos a su servidor.

14.2 SessionStorage

El sistema sessionStorage reemplaza las cookies de sesión. Tanto las cookies como sessionStorage, mantienen los datos disponibles durante un período específico de tiempo, pero mientras las cookies usan el navegador como referencia, sessionStorage usa solo una ventana o incluso pestaña. Esto significa que las cookies creadas para una sesión estarán disponibles mientras que la ventana del navegador continúe abierta, pero los datos creados con sessionStorage estarán solo disponibles mientras no sea cerrada la ventana que los creó.

14.2.1 Implementar un sistema de almacenamiento de datos

Debido a que ambos sistemas, sessionStorage y localStorage, trabajan con la misma interfaz, vamos a necesitar solo un documento HTML y un sencillo formulario para probar los códigos y experimentar con esta API:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title> API Web Storage</title>
  <link rel="stylesheet" href="storage.css">
<script src="storage.js"></script>
</head>
<body>
  <section id="cajaformulario">
    <form name="formulario">
      <label for="keyword">Clave: </label><br>
      <input type="text" name="clave" id="clave"><br>
      <label for="text">Valor: </label><br>
      <textarea name="text" id="texto"></textarea><br>
      <input type="button" id="grabar" value="Grabar"
    </form>
  </section>
  <section id="cajadatos">
    No hay información disponible
  </section>
</body>
</html>
```

Código 14-1

Plantilla para la API Web Storage.

También crearemos un grupo de reglas de estilo simples para dar forma a la página y diferenciar el área del formulario de la caja donde se mostrarán los datos:

```
#cajaformulario{
  float: left;
  padding: 20px;
  border: 1px solid #999999;
}
#cajadatos{
  float: left;
  width: 400px;
  margin-left: 20px;
  padding: 20px;
  border: 1px solid #999999;
}
#clave, #texto{
  width: 200px;
}
#cajadatos >
  div{
  padding: 5px;
  border-bottom: 1px solid #999999;
}
```

Código 14-2

Estilos para nuestra plantilla.

Hágalo usted mismo

Cree un archivo HTML con el código del Código 14-1 y un archivo CSS llamado storage.css con los estilos del Código 14-2. También necesitará crear un archivo llamado storage.js para grabar y probar los códigos Javascript presentados a continuación.

14.2.2 Crear datos

sessionStorage y localStorage almacenan datos como ítems. Éstos están formados por una pareja de clave y valor, y cada valor es convertido en un texto antes de ser almacenado. Imagine que los ítems son variables, cada uno con un nombre y un valor, que pueden ser creadas, modificadas o eliminadas. Esta api ofrece métodos específicos para crear y obtener un ítem del espacio de almacenamiento:

setItem(clave, valor): Éste es el método que tenemos que llamar para crear un ítem. El ítem será creado con una clave y un valor de acuerdo a los atributos especificados. Si ya existe un ítem con la misma clave, su valor será actualizado, por lo que este método puede utilizarse también para modificar datos almacenados previamente.

getItem(clave): Para obtener el valor de un ítem, debemos llamar a este método especificando la clave del ítem que queremos leer. La clave en este caso es la misma que declaramos cuando creamos al ítem con setItem().

```
function iniciar(){
  var boton=document.getElementById('grabar');
  boton.addEventListener('click', nuevoitem);
}
function nuevoitem(){
  var clave = document.getElementById('clave').value;
  var valor = document.getElementById('texto').value;
  sessionStorage.setItem(clave,valor);
  mostrar(clave);
}
function mostrar(clave){
  var cajadatos = document.getElementById('cajadatos');
  var valor = sessionStorage.getItem(clave);
  cajadatos.innerHTML = '<div>' + clave + ' - ' + valor + '</div>';
}
addEventListener('load', iniciar);
```

Código 14-3

Almacenando y leyendo datos.

El proceso es extremadamente simple. Los métodos son parte de sessionStorage y son llamados con la sintaxis sessionStorage.setItem(). En el Código 14-3, la función nuevoitem() es ejecutada cada vez que el usuario hace clic en el botón del formulario. Esta función crea un ítem con la información insertada en los campos del formulario y luego llama a la función mostrar(). Esta última función devuelve el ítem de acuerdo al método getItem() y muestra el valor de su clave en la pantalla.

Además de estos métodos, la API también ofrece una sintaxis abreviada para crear y leer ítems desde el espacio de almacenamiento. Podemos usar la clave del ítem como una propiedad de sessionStorage para obtener el valor correspondiente.

Como sucede con cualquier otra propiedad, hay dos sintaxis posibles. Podemos usar una variable para la clave, encerrada entre corchetes (por ejemplo, sessionStorage[clave]=valor) o podemos usar directamente el nombre de la propiedad (por ejemplo, sessionStorage.miitem=valor).

```
function iniciar(){
  var boton = document.getElementById('grabar');
  boton.addEventListener('click', nuevoitem);
}
function nuevoitem(){
  var clave =
document.getElementById('clave').value;
  var valor = document.getElementById('texto').value;
  sessionStorage[clave] = valor;
  mostrar(clave);
}
function mostrar(clave){
  var cajadatos = document.getElementById('cajadatos');
  var valor = sessionStorage[clave];
  cajadatos.innerHTML = '<div>' + clave + ' - ' + valor + '</div>';
}
addEventListener('load', iniciar);
```

Código 14-4

Usando un atajo para trabajar con ítems.

Hágalo usted mismo

Aproveche los conceptos estudiados de la API Forms en el Capítulo 5 para controlar la validez de los campos del formulario y no permitir la inserción de ítems vacíos o inválidos.

14.2.3 Leer datos

El ejemplo anterior solo devuelve el último ítem grabado. Vamos a mejorar este código aprovechando otros métodos y propiedades provistos por la API con el propósito de manipular ítems:

length: Esta propiedad devuelve el número de ítems guardados en el espacio de almacenamiento de la aplicación. Trabaja exactamente como la propiedad length usada en Javascript para procesar matrices, y es útil para lecturas secuenciales.

key(índice): Los ítems son almacenados secuencialmente, enumerados con un índice automático que comienza en el valor 0. Con este método podemos leer un ítem específico o crear un bucle para obtener toda la información almacenada.

```
function iniciar(){
  var boton = document.getElementById('grabar');
  boton.addEventListener('click', nuevoitem);
  mostrar();
}
function nuevoitem(){
  var clave = document.getElementById('clave').value;
  var valor = document.getElementById('texto').value;

  sessionStorage.setItem(clave,valor);
  document.getElementById('clave').value='';
  document.getElementById('texto').value='';
  mostrar();
}
function mostrar(){
  var cajadatos = document.getElementById('cajadatos');
  cajadatos.innerHTML='';
  for(var f = 0; f < sessionStorage.length; f++){
var clave = sessionStorage.key(f);
var valor = sessionStorage.getItem(clave);
cajadatos.innerHTML += '<div>' + clave + ' - ' + valor + '</div>';
  }
}
addEventListener('load', iniciar);
```

Código 14-5

Lista de ítems en el espacio de almacenamiento.

El propósito del Código 14-5 es mostrar una lista de ítems en la caja derecha de la pantalla. La función mostrar() fue mejorada usando la propiedad length y el método key().

Dentro de un bucle for, es llamado el método key(), que retornará la clave de cada ítem. Vamos a poner un ejemplo: si el ítem en la posición 0 del espacio de almacenamiento es creado con la clave miitem, entonces la instrucción sessionStorage.key(0) retornará el valor miitem. Llamando a este método desde un bucle permite listar todos los ítems en la pantalla, con sus correspondientes claves y valores. La función mostrar() es llamada desde el final de la función iniciar(), en el mismo momento en que se carga la aplicación, para mostrar los ítems que fueron grabados previamente en el espacio de almacenamiento.

14.2.4 Eliminar datos

Los ítems pueden ser creados, leídos y, por supuesto, eliminados. Hay dos métodos que pueden ser utilizados para este propósito:

removeItem(clave): Elimina un ítem individual. La clave para identificar el ítem debe ser la misma que fue declarada cuando el ítem fue creado con el método setItem().

clear(): Vacía el espacio de almacenamiento, es decir, elimina todos los ítems.

```
function iniciar(){
  var boton = document.getElementById('grabar');
  boton.addEventListener('click', nuevoitem);
  mostrar();
}
function nuevoitem(){
  var clave = document.getElementById('clave').value;
  var valor = document.getElementById('texto').value;

  sessionStorage.setItem(clave,valor);
  document.getElementById('clave').value='';
  document.getElementById('texto').value='';
  mostrar();
}
function mostrar(){
  var cajadatos = document.getElementById('cajadatos');
  cajadatos.innerHTML = '<div><input type="button"
 onclick="eliminarTodo()" value="Eliminar Todo"></div>';
  for(var f = 0;f < sessionStorage.length; f++){
    var clave = sessionStorage.key(f);
    var valor = sessionStorage.getItem(clave);
    cajadatos.innerHTML += '<div>' + clave + ' - ' + valor +
 '<br><input type="button" onclick="eliminar(\'' + clave + '\')"
 value="Eliminar"></div>';
  }
} function eliminar(clave){
  if(confirm("¿Está Seguro?")){
    sessionStorage.removeItem(clave);
    mostrar();
  }
}
function removeAll(){
if(confirm('Are you sure?')){
sessionStorage.clear();
show();
}
}
addEventListener('load', initiate);
```

Código 14-6

Eliminar ítems.

Las funciones iniciar() y nuevoitem() del Código 14-6 son las mismas del código anterior. Solo cambia la función mostrar() para incorporar el atributo de eventos onclick para llamar a las funciones que eliminarán un ítem en particular o todos ellos. El código crea un botón Eliminar para cada ítem de la lista y, además, un botón ubicado en la cabecera de la lista, que permite eliminar todos los ítems.

Las funciones eliminar() y eliminarTodo() se encargan respectivamente de eliminar el ítem seleccionado y limpiar el espacio de almacenamiento. Cada función llama a la función mostrar() para actualizar la lista de ítems en pantalla.

Hágalo usted mismo

Con el Código 14-6 podrá comprobar la forma en la que la información es procesada por sessionStorage. Abra la plantilla del Código 14-1 en su navegador, cree nuevos ítems y luego abra la plantilla en una nueva ventana. La información en cada ventana será diferente. Mientras la primera ventana conserva su información, el espacio de almacenamiento de la nueva ventana estará vacío. A diferencia de otros sistemas (como las cookies), para sessionStorage cada ventana es considerada una instancia diferente de la aplicación y la información de la sesión no se comparte entre ellas.

El sistema SessionStorage conserva los datos creados en una ventana solo hasta que esta ventana es cerrada. Es útil para controlar carros de compra o cualquier otra aplicación que requiera acceso a datos por cortos períodos de tiempo.

14.3 LocalStorage

Disponer de un sistema confiable para almacenar datos durante la sesión de una ventana puede ser extremadamente útil en algunas circunstancias, pero un sistema de almacenamiento temporal no es suficiente cuando intentamos simular poderosas aplicaciones de escritorio en la Web. Para cubrir estas necesidades, la API Storage ofrece otro sistema que reserva un espacio de almacenamiento para cada aplicación (cada origen) y mantiene la información disponible permanentemente. Con LocalStorage, podemos grabar largas cantidades de datos y dejar que el usuario decida si la información es útil y debe ser conservada o no. El sistema usa la misma interfaz que SessionStorage, así que cada uno de los métodos y propiedades estudiados hasta el momento en este capítulo también están disponibles para LocalStorage. Solo hay que la sustituir el prefijo session por local.

```
function iniciar(){
  var boton=document.getElementById('grabar');
  boton.addEventListener('click', nuevoitem, false);
  mostrar();
}
function nuevoitem(){
  var clave=document.getElementById('clave').value;
  var valor=document.getElementById('texto').value;
  localStorage.setItem(clave,valor);
  mostrar();
  document.getElementById('clave').value='';
  document.getElementById('texto').value='';
}
function mostrar(){
  var cajadatos=document.getElementById('cajadatos');
  cajadatos.innerHTML='';
  for(var f=0;f<localStorage.length;f++){
    var clave=localStorage.key(f);
    var valor=localStorage.getItem(clave);
    cajadatos.innerHTML+='<div>'+clave+' - '+valor+'</div>';
  }
}
addEventListener('load', iniciar);
```

Código 14-7

Usar localStorage.

En el Código 14-7, simplemente reemplazamos sessionStorage por localStorage en el código de uno de los ejemplos anteriores. Ahora, cada ítem creado será conservado a través de diferentes ventanas e incluso luego de que todas las ventanas del navegador sean cerradas.

Hágalo usted mismo

Usando el documento HTML del Código 14-1, pruebe el Código 14-7. Este código creará un nuevo ítem con la información insertada en el formulario y automáticamente listará todos los ítems disponibles en el espacio de almacenamiento reservado para esta aplicación. Cierre el navegador y abra el archivo HTML nuevamente. La información es preservada, por lo que podrá ver aún en pantalla todos los ítems ingresados previamente.

14.3.1 Evento storage

Debido a que localStorage hace que la información esté disponible en cada ventana en la que la aplicación haya sido cargada, surgen al menos dos problemas: debemos resolver cómo se comunicarán entre sí las ventanas y cómo haremos para mantener la información actualizada en cada una de ellas. En respuesta a ambos problemas, la especificación incluye el evento storage.

storage: Este evento será disparado por la ventana cada vez que ocurra un cambio en el espacio de almacenamiento. Puede ser usado para informar a cada ventana abierta con la misma aplicación que los datos han cambiado en el espacio de almacenamiento y que se debe hacer algo al respecto.

```
function iniciar(){
  var boton=document.getElementById('grabar');
  boton.addEventListener('click', nuevoitem);
  addEventListener("storage", mostrar);
  mostrar();
}
function nuevoitem(){
  var clave = document.getElementById('clave').value;
  var valor = document.getElementById('texto').value;

  localStorage.setItem(clave,valor);
  document.getElementById('clave').value='';
  document.getElementById('texto').value='';
  mostrar();
}

function mostrar(){
  var cajadatos = document.getElementById('cajadatos');
  cajadatos.innerHTML = '';
  for(var f = 0; f < localStorage.length; f++){
    var clave = localStorage.key(f);
    var valor = localStorage.getItem(clave);
    cajadatos.innerHTML += '<div>' + clave + ' - ' + valor + '</
 div>';
  }
}
addEventListener('load', iniciar);
```

Código 14-8

Detectar el evento storage para mantener la lista de ítems actualizada.

En este ejemplo la función mostrar() detecta el evento comenzar a escuchar al evento storage cada vez que un ítem es creado, actualizado o borrado. Ahora, si algo cambia en una ventana, el cambio será mostrado automáticamente en el resto de las ventanas que están ejecutando la misma aplicación.

Hágalo usted mismo

Usando el documento HTML del Código 14-1, pruebe el Código 14-8. Recuerde cargar antes todos los archivos al servidor. Abra el documento en dos ventanas diferentes, inserte o actualice un ítem en una venteana y regrese a la otra ventana para comprobar cómo funciona el evento storage.

Este código creará un nuevo ítem con la información insertada en el formulario y automáticamente listará todos los ítems disponibles en el espacio de almacenamiento reservado para esta aplicación. Cierre el navegador y abra el archivo HTML nuevamente. La información es preservada, por lo que podrá ver aún en pantalla todos los ítems ingresados previamente.

El evento storage devuelve un objeto con varias propiedades para proporcionar información sobre los cambios producidos en el espacio de almacenamiento:

key: Esta propiedad devuelve la clave del ítem modificado.

oldValue: Esta propiedad devuelve el valor previo del ítem modificado.

newValue: Esta propiedad devuelve el nuevo valor asignado al ítem.

url: Esta propiedad devuelve la URL del documento que produjo la modificación. Recuerde que el espacio de almacenamiento será reservado para el dominio completo, de modo que diferentes documentos tendrán acceso a él.

storageArea: Esta propiedad devuelve una matriz que contiene todas las parejas de claves y los valores del espacio de almacenamiento después de la modificación.

```
function iniciar(){
  var boton = document.getElementById('grabar');
  boton.addEventListener('click', nuevoitem);
  addEventListener("storage", storagechange);
  mostrar();
}
function nuevoitem(){
  var clave = document.getElementById('clave').value;
  var valor = document.getElementById('texto').value;
  localStorage.setItem(clave,valor);
  document.getElementById('clave').value='';
  document.getElementById('texto').value='';
  mostrar();
}

function storagechange(e){
  console.log(e.key);
  console.log(e.oldValue);
  console.log(e.newValue);
  console.log(e.url);
  console.log(e.storageArea);
  mostrar();
}

function mostrar(){
  var cajadatos = document.getElementById('cajadatos');
  cajadatos.innerHTML = '';
  for(var f = 0; f < localStorage.length; f++){
    var clave = localStorage.key(f);
    var valor = localStorage.getItem(clave);
    cajadatos.innerHTML += '<div>' + clave + ' - ' + valor + '</
                           div>';
  }
}
addEventListener('load', iniciar);
```

Código 14-9

Acceder a las propiedades de eventos.

En el Código 14-9 añadimos una nueva función para controlar el evento storage. La función almacenarcambios es llamada por el evento cuando el espacio de almacenamiento es modificado por otra ventana. Muestra los valores de las propiedades, una por una, en la ventana del navegador y al final llama a la función mostrar()para actualizar la información en pantalla.

15 API IndexedDB

15.1 Una API de bajo nivel

La API estudiada en el capítulo anterior es útil para el almacenamiento de pequeñas cantidades de datos, pero cuando se trata de grandes cantidades de datos estructurados, debemos recurrir a un sistema de base de datos. La API IndexedDB es la solución provista por HTML5 para resolver estas situaciones.

IndexedDB es un sistema de base de datos destinado a almacenar información indexada en el ordenador del usuario. Fue desarrollada como una API de bajo nivel con la intención de permitir un amplio espectro de usos. Esto la convierte en una de las API más poderosas de todas, pero también una de las más complejas. Su objetivo es proveer la estructura más básica posible para permitir a los desarrolladores construir librerías y crear interfaces de alto nivel para satisfacer necesidades específicas.

En una API de bajo nivel como ésta tenemos que hacernos cargo de cada aspecto y controlar las condiciones de cada proceso en cada operación realizada. El resultado es una API a la que la mayoría de los desarrolladores tardará en acostumbrarse y probablemente la utilizarán de forma indirecta a través de otras librerías populares como jQuery u otras que surgirán en un futuro cercano. La estructura propuesta por IndexedDB es también diferente de SQL u otros sistemas de bases de datos populares. La información es almacenada en la base de datos como objetos (registros) dentro de almacenes de objetos (tablas). Los almacenes de objetos no tienen una estructura específica, solo un nombre e índices para poder encontrar los objetos en su interior. Estos objetos tampoco tienen una estructura predefinida, pueden ser diferentes unos de otros, y tan complejos como necesitemos. La única condición para los objetos es que contengan al menos una propiedad declarada como índice para que el almacén de objetos pueda encontrarlos.

15.1.1 Base de datos

La base de datos es sencilla. Como cada base de datos está asociada a un ordenador y un sitio web o aplicación, no hay que asignar usuarios ni gestionar otras bases de datos. Solo hay que especificar el nombre y la versión, y la base de datos está lista. Veamos dos métodos fudamentales para el trabajo con bases de datos:

open(nombre, versión): Este método crea o abre la base de datos con el nombre y la versión especificados en sus atributos. La versión puede ser ignorada al crear la base de datos, pero es necesaria para establecer una nueva versión para una base de datos existente. El método devuelve un objeto que dispara dos eventos (success y error) que indican respectivamente el éxito o los errores surgidos durante el proceso de creación o acceso a la base de datos.

deleteDatabase(nombre): Este método borra la base de datos cuyo nombre ha sido especificado en el atributo.

La API permite asignar una versión a la base de datos. Cuando es necesario actualizar la estructura de una base de datos en el lado del servidor para agregar más tablas o índices, normalmente se crea una nueva base de datos con la estructura requerida y luego se migra la información hacia la nueva estructura. Este proceso requiere acceso absoluto al servidor e incluso la capacidad de encenderlo y apagarlo. Sin embargo, no podemos apagar el ordenador del usuario para realizar el mismo proceso. Para manejar esta situación, hay que declarar una nueva versión en el método open(), que permiten migrar la información de la versión anterior a la nueva.

Para trabajar con versiones de bases de datos, la API ofrece el evento upgradeneeded. Este evento se dispara cuando el método open() intenta abrir una base de daos inexistente o una nueva versión de la base de datos especificada.

15.1.2 Objetos y Almacenes de objetos

Lo que solemos llamar registros, en IndexedDB son llamados objetos. Los objetos incluyen propiedades para almacenar e identificar valores. La cantidad de propiedades y la forma en la que los objetos son estructurados es irrelevante. Solo deben incluir al menos una propiedad declarada como índice para que el almacén de objetos pueda reconocerlos.

Los almacenes de objetos (tablas) tampoco tienen una estructura específica. Solo deben ser declarados el nombre y uno o más índices en el momento de su creación, para que más tarde puedan ser encontrados los objetos en su interior.

Como podemos ver en la Figura 15-1, un almacén de objetos contiene diversos objetos con diferentes propiedades. En este ejemplo, algunos objetos tienen la propiedad DVD, otros tienen la propiedad Libro, etc. Cada uno tiene su propia estructura, pero todos deberán tener al menos una propiedad declarada como índice para que el almacén de objetos pueda encontrarlos. En el ejemplo de la Figura 15-1, este índice podría ser la propiedad Id.

Almacén de Objetos
Objeto 1
Id
Nombre
DVD
Vídeo
Objeto 2
Id
Nombre
DVD
Objeto 3
Id
Nombre
DVD
Libro
Objeto 4
Id
Nombre
Libro

Figura 15-1

Objetos con diferentes propiedades almacenados en un almacén de objetos.

Para trabajar con objetos y almacenes de objetos necesitamos crear el almacén de objetos, declarar las propiedades que serán usadas como índices y luego comenzar a almacenar objetos. No es necesario pensar en la estructura o el contenido de los objetos en este momento, solo considerar los índices que vamos a utilizar para encontrarlos más adelante. La API proporciona varios métodos para manipular almacenes de objetos:

createObjectStore(nombre, objeto): Este método crea un nuevo almacén de objetos con el nombre y la configuración declarada por sus atributos. El atributo nombre es obligatorio. El atributo objeto es un objeto que puede incluír las propiedades clave y autoincremento. La propiedad clave declarará un índice común para todos los objetos mientras la propiedad autoIncremento es un valor booleano que determina si el almacén de objetos tendrá un generador de claves automático.

deleteObjectStore(nombre): Este método elimina el almacén de objetos al que corresponde el nombre declarado por el atributo nombre.

objectStore(nombre): Para acceder a los objetos en un almacén de objetos, debe ser iniciada una transacción y el almacén de objetos debe ser abierto para esa transacción. Este método abre el almacén de objetos con el nombre declarado por el atributo nombre.

Los métodos createObjectStore() y deleteObjectStore(), así como otros métodos responsables de la configuración de la base de datos, solo pueden ser aplicados cuando la base de datos es creada o mejorada en una nueva versión.

15.1.3 Índices

Para encontrar objetos en un almacén de objetos necesitamos declarar algunas propiedades de estos objetos como índices. Una forma fácil de hacerlo es declarar la propiedad clave en el método createObjectStore(). La propiedad declarada como clave será un índice común para cada objeto almacenado en ese almacén de objetos particular. Cuando declaramos el atributo clave, esta propiedad debe estar presente en todos los objetos.

Además del atributo clave podemos declarar todos los índices que necesitemos para un almacén de objetos usando métodos especiales provistos para este propósito:

createIndex(nombre, propiedad, objeto): Este método crea un índice para un almacén de objetos específico. El atributo nombre es un nombre con el que identificar al índice, el atributo propiedad es la propiedad que será usada como índice y el atibuto objeto es un atributo que puede incluir las propiedades único o multiEntrada: La propiedad único es un valor booleano que indica si el valor del índice debe ser único o si existe la posibilidad de que dos o más objetos compartan el mismo valor para el mismo índice. La propiedad multiIndice determina cómo se generarán las claves de un objeto cuando el índice es una matriz.

deleteIndex(nombre): Si ya no necesitamos un índice, podemos eliminarlo usando este método. Ese método solo puede ser llamado desde una transacción versionchange.

index(nombre): Para usar un índice primero tenemos que crear una referencia al índice y luego asignar esta referencia a la transacción. El método index() crea esta referencia para el índice declarado en el atributo nombre.

15.1.4 Transacciones

Un sistema de base de datos que trabaja en un navegador debe contemplar algunas circunstancias únicas que se dan en otras plataformas. El navegador puede fallar, puede ser cerrado abruptamente, el proceso puede ser detenido por el usuario, o simplemente otro sitio web puede ser cargado en la misma ventana, por ejemplo. Existen muchas situaciones en las que trabajar directamente con la base de datos puede causar un mal funcionamiento o incluso corromper los datos. Para prevenir que algo así suceda, cada acción es realizada por medio de transacciones.

El método que genera una transacción se llama transaction(). Para declarar el tipo de transacción, contamos con los siguientes tres atributos:

readonly: Este atributo genera una transacción de solo lectura y no permite modificaciones.

readwrite: Usando este tipo de transacción podemos leer y escribir. Permite modificaciones pero no es posible añadir o eliminar almacenes de objetos o índices.

versionchange: Este tipo de transacción solo es utilizada para actualizar la versión de la base de datos o para eliminar almacenes de objetos e índices.

Las transacciones más comunes son las transacciones de lectura y escritura (readwrite). Sin embargo, para evitar un uso inadecuado, se establece por defecto

el tipo readonly (solo lectura). Así, cuando solo necesitamos obtener información de la base de datos, lo único que debemos hacer es declarar el destino de la transacción (normalmente el nombre del almacén de objetos de donde vamos a leer esta información) y el tipo de transacción se establece de forma automática.

15.1.5 Métodos de almacenes de objetos

Para interactuar con almacenes de objetos, o leer y almacenar información, la API provee varios métodos:

add(objeto): Este método recibe una pareja de clave y valor o un objeto que contiene varias parejas de clave y valor, y genera con los datos provistos un objeto que es agregado al almacén de objetos seleccionado. Si ya existe un objeto con el mismo valor de índice, el método add() devuelve un error.

put(objeto): Este método es similar al anterior, pero éste sobreescribe un objeto preexistente con el mismo índice. Es útil para modificar un objeto ya almacenado en el almacén de objetos seleccionado.

get(clave): Podemos obtener un objeto específico del almacén de objetos usando este método. El atributo clave es el valor del índice del objeto que queremos leer.

delete(clave): Para eliminar un objeto del almacén de objetos seleccionado solo tenemos que llamar a este método con el valor del índice del objeto a eliminar como atributo.

15.2 Implementar IndexedDB

¡Demasiada la teoría! Es momento de crear nuestra primera base de datos y aplicar algunos de los métodos ya mencionados en este capítulo. Vamos a simular una aplicación para almacenar información sobre películas.

Importante

Los nombres de las propiedades (id, nombre y fecha) son los que vamos a utilizar para nuestros ejemplos en el resto del capítulo. La información fue recolectada del sitio web www.imdb.com, pero usted puede utilizar su propia lista o información al azar para probar los códigos.

Puede agregar a la base los datos que usted desee, pero para nuestros ejemplos usaremos los siguientes:

id: tt0068646 nombre: El padrino fecha: 1972

id: tt0086567 nombre: Juegos de guerra fecha: 1983

id: tt0111161 nombre: Cadena perpetua fecha: 1994

id: tt1285016 nombre: La red social fecha: 2010

15.2.1 Plantilla

Como siempre, necesitamos un documento HTML y algunos estilos CSS para crear las cajas en pantalla que contendrán el formulario apropiado y la información retornada. El formulario nos permitirá insertar nuevas películas dentro de la base de datos solicitándonos una clave, el título y el año en el que la película fue realizada.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title> API IndexedDB</title>
  <link rel="stylesheet" href="indexed.css">
 <script src="indexed.js"></script>
</head>
<body>
  <section id="cajaformulario">
    <form name="formulario">
      <label for="clave">Clave: </labe><br>
      <input type="text" name="clave" id="clave"><br>
      <label for="text" name=Título: </label><br>
      <input type="text" name="texto" id="texto"><br>
      <label for="year" name=Año: </label><br>
      <input type="text" name="fecha" id="fecha"><br>
      <input type="button" id="grabar" value="Grabar">
    </form>
  </section>
  <section id="cajadatos">
    No hay información disponible
  </section>
</body>
</html>
```

Código 15-1

Plantilla para IndexedDB API.

Los estilos CSS definen las cajas para el formulario y la forma en la que se mostrará la información en pantalla:

```
#cajaformulario{
  float: left;
  padding: 20px;
  border: 1px solid #999999;
}
#cajadatos{
  float: left;
  width: 400px;
  margin-left: 20px;
  padding: 20px;
  border: 1px solid #999999;
}
#clave, #texto{
  width: 200px;
} #

cajadatos > div{
  padding: 5px;
  border-bottom: 1px solid #999999;
}
```

Código 15-2

Estilos para las cajas.

Hágalo usted mismo

Necesitará un archivo HTML para la plantilla del Código 15-1, un archivo CSS llamado indexed.css para los estilos del Código 15-2 y un archivo Javascript llamado indexed.js para introducir todos los códigos estudiados a continuación.

15.2.2 Abrir la base de datos

Lo primero que debemos hacer en el código Javascript es abrir la base de datos. El atributo indexedDB y el método open() abren la base con el nombre declarado o crean una nueva si no existe:

```
var cajadatos, bd
function iniciar(){
  cajadatos = document.getElementById('cajadatos');
  var boton = document.getElementById('grabar');
  boton.addEventListener('click', agregarobjeto);

  var solicitud=indexedDB.open('mibase');
  solicitud.addEventListener('error', errores);
  solicitud.addEventListener('success', crear);
  solicitud.addEventListener('upgradeneeded', crearbd);
}
```

Código 15-3

Abrir la base de datos.

La función iniciar() del Código 15-3 prepara los elementos de la plantilla y abre la base de datos. La instrucción indexedDB.open() intenta abrir la base de datos con el nombre mibase y retorna el objeto solicitud con el resultado de la operación. Los eventos error, success o upgradeneeded son disparados sobre este objeto en caso de error, éxito, o de que sea necesaria una actualización, respectivamente.

Los eventos son una parte importante de esta API. IndexedDB es una API síncrona y asíncrona. La parte síncrona está siendo desarrollada en estos momentos y está destinada a trabajar con la API Web Workers. En cambio, la parte asíncrona está destinada a un uso web normal y ya se encuentra disponible. Un sistema asincrono realiza tareas en segundo plano y retorna los resultados posteriormente. Con este propósito, esta API dispara diferentes eventos en cada operación. Cada acción sobre la base de datos y su contenido es procesada en segundo plano (mientras el sistema ejecuta otros códigos) y los eventos correspondientes son disparados luego para informar de los resultados obtenidos.

Luego de que la API procesa la solicitud para la base de datos, es disparado un evento error o success de acuerdo al resultado y una de las funciones errores() o crear() es ejecutada para controlar los errores o continuar con la definición de la base de datos, respectivamente.

```
function errores(e){
alert('Error: '+e.code+' '+e.message);
}
function crear(e){
   bd=e.target.result;
}
```

Código 15-4

Declarando la versión y respondiendo a eventos.

Las funciones errores() y crear() son muy sencillas (no necesitamos hacer nada más que procesar errores u obtener referencias de la base de datos para esta aplicación de muestra). La información sobre los errores se muestra usando las propiedades code y message del evento, y luego se obtiene una referencia a la base de datos usando la propiedad result, que es almacenada en la variable global bd. Esta variable bd será utilizada para representar la base de datos en el resto del código.

result: Esta propiedad devuelve un objeto con el resultado de la consulta. Por ejemplo, un objeto que representa la base de datos o un objeto que representa el objeto devuelto desde el almacén de objetos.

source: devuelve un objeto con información sobre la fuente de la solicitud.

transaction: devuelve un objeto con información sobre la transacción.

readyState: devuelve la condición actual de la transacción. Los valores posibles son pending (pendiente) o done (hecho).

error: devuelve el error de la solicitud.

En nuestros ejemplos usaremos la propiedad result para obtener una referencia de la base de datos y de los objetos del almacén de objetos.

15.2.3 Almacenes de objetos e índices

Si se declara una nueva versión de la base de datos mediante el método open(), o si la base de datos no existe aún, se dispara el evento upgradeneeded y se llama la función createbd() para controlar el evento. Llegados a este punto, seguramente se habrá preguntado ya qué clase de objetos necesita almacenar en la base de datos y cómo va a obtener más adelante la información contenida en los almacenes de objetos. Si la estructura no es correcta o si quiere añadir algo a la configuración de la base de datos en el futuro, deberá declarar una nueva versión y migrar los datos desde la anterior, o modificar la estructura a través de una transacción versionchange.

```
function creabd(e){

var bd = e.target.result

var almacen=bd.createObjectStore("peliculas",{keyPath: "id"});

almacen.createIndex("BuscarFecha", "fecha",{unique: false});

}
```

Código 15-5

Declarar almacenes de objetos e índices.

Para nuestro ejemplo solo necesitamos un almacén de objetos (para almacenar películas) y dos índices. El primer índice, id, es declarado como el atributo clave para el método createObjectStore() al crear el almacén de objetos. El segundo índice es asignado al almacén de objetos mediante el método createIndex(). Este índice ha sido identificado con el nombre BuscarFecha y declarado para la propiedad fecha. Más adelante vamos a usar este índice para ordenar películas por año.

Observe que en esta función, de nuevo hemos tenido que tomar la referencia a la base de datos de la propiedad result. Cuando se crea la función createbd(), el evento success no ha sido activado aún y la referencia a la base de datos aún no ha sido creada o, lo que es lo mismo, el valor de la variable bd aún no ha sido definido.

15.2.4 Agregar objetos

De momento tenemos una base de datos con el nombre mibase y un almacén de objetos llamado peliculas con dos índices: id y fecha. Con esto ya podemos comenzar a agregar objetos al almacén:

```
function agregarobjeto(){
  var clave=document.getElementById('clave').value;
  var titulo=document.getElementById('texto').value;
  var fecha=document.getElementById('fecha').value;

  var transaccion=bd.transaction(['peliculas'],"readwrite");
  var almacen=transaccion.objectStore('peliculas');
  var solicitud=almacen.add({id: clave, nombre: titulo, fecha:
                                    fecha});
  solicitud.addEventListener('success', function(){mostrar(clave) },
                                    false);
document.getElementById('clave').value='';
document.getElementById('texto').value='';
document.getElementById('fecha').value='';
}
```

Código 15-6

Agregar objetos.

Al comienzo de la función iniciar() habíamos agregado al botón del formulario un detector del evento click. Este detector ejecuta la función agregarobjeto() cuando el evento es disparado. Esta función toma los valores de los campos del formulario (clave, texto y fecha) y luego genera una transacción para almacenar un nuevo objeto usando esta información.

Para comenzar la transacción, debemos usar el método transaction(), especificar el almacén de objetos involucrado en la transacción y el tipo de transacción. En este caso el único almacén es peliculas y el tipo es declarado como readwrite. El próximo paso es seleccionar el almacén de objetos que vamos a usar. Debido a que la transacción puede ser originada para varios almacenes de objetos, tenemos que declarar cuál corresponde a la operación que queremos ejecutar. Mediante el método objectStore(), abrimos el almacén de objetos y lo asignamos a la transacción con esta línea: transaccion.objectStore(’peliculas’).

Es momento de agregar el objeto al almacén de objetos. En este ejemplo usamos el método add() debido a que queremos crear un nuevo objeto, pero podríamos haber utilizado el método put() en su lugar si nuestra intenc ión hubiese sido modificar o reemplazar un objeto anterior. El método add() usa las propiedades id, nombre y fecha, y las variables clave, titulo y fecha, y crea un objeto que usa estos valores como clave y valor respectivamente.

Finalmente, la función mostrar() detecta el evento success disparado por la solicitud y se ejecuta en caso de éxito. Existe también un evento error, por

supuesto, pero como la respuesta a los errores depende de las necesidades de la aplicación, no consideramos esta posibilidad en este ejemplo.

15.2.5 Leer objetos

Si el objeto es correctamente almacenado, se dispara el evento success y se ejecuta la función mostrar(). En el Código 15-6, esta función se declara dentro de una función anónima para poder pasar la variable clave. Ahora vamos a tomar este valor para leer el objeto previamente almacenado:

```
function mostrar(clave){
  var transaccion=bd.transaction(['peliculas']);
  var almacen=transaccion.objectStore('peliculas');
  var solicitud=almacen.get(clave);
  solicitud.addEventListener('success', mostrarlista);
}
function mostrarlista(e){
  var resultado=e.target.result;
  cajadatos.innerHTML='<div>'+resultado.id+' - '+resultado.nombre+'-
                      '+resultado.fecha+'</div>';
}
```

Código 15-7

Leer y mostrar el objeto almacenado.

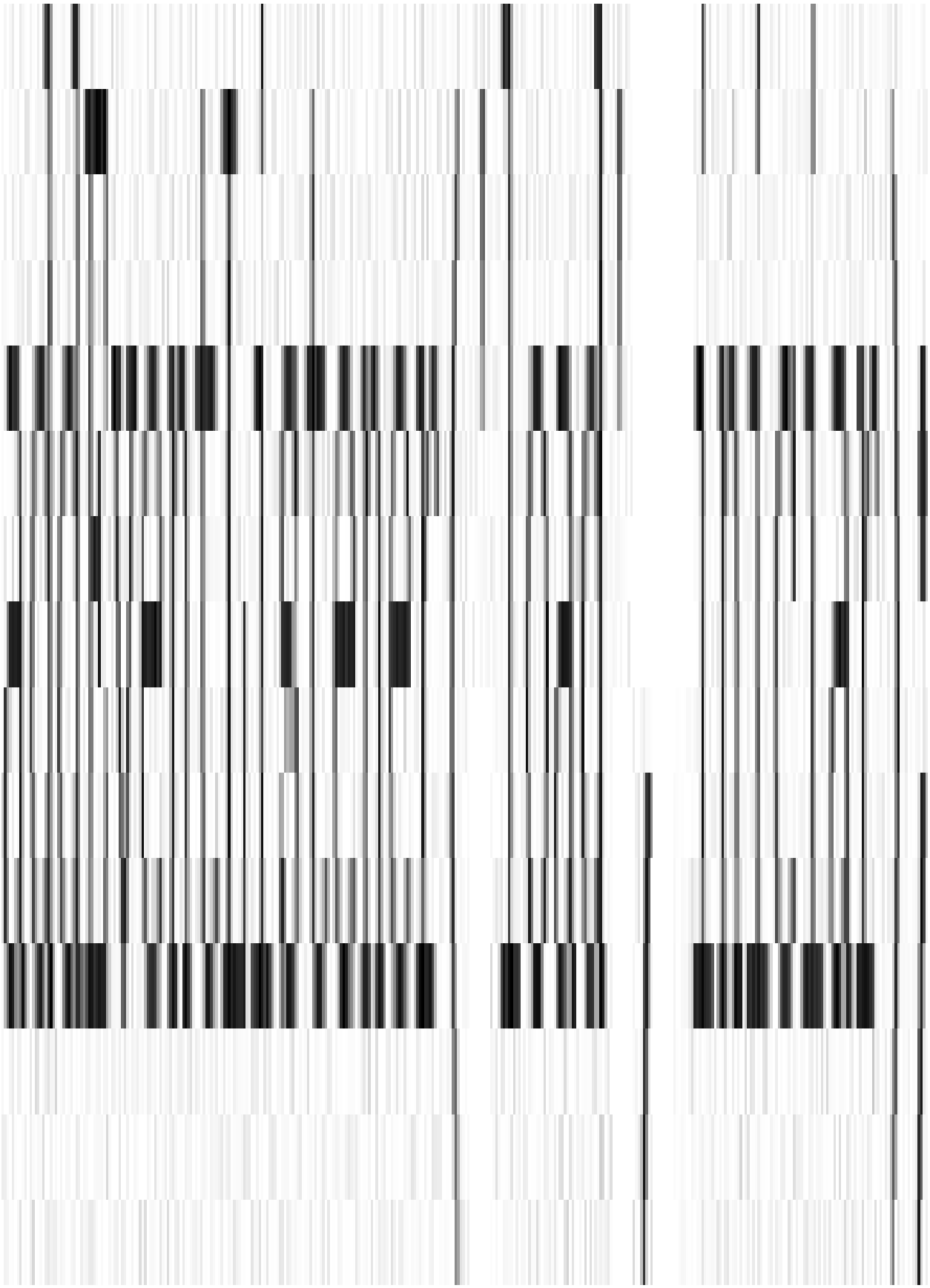
El Código 15-7 genera una transacción readonly y usa el método get() para leer el objeto con la clave recibida. No tenemos que declarar el tipo de transacción porque readonly es establecido por defecto. El método get() devuelve el objeto almacenado con la propiedad id = clave.

Si, por ejemplo, insertamos la película El Padrino de nuestra lista, la variable clave tendrá el valor tt0068646. Este valor es recibido por la función mostrar() y usado por el método get() para leer la película El Padrino. Como puede ver, este código es solo ilustrativo ya que solamente puede retornar la misma película que acabamos de almacenar.

Como cada operación es asíncrona, necesitamos dos funciones para mostrar la información. La función mostrar(), en primer lugar, genera la transacción y más adelante la función mostrarlista() muestra los valores de las propiedades del objeto en pantalla, en caso de éxito. Otra vez, unicamente estamos detectando el evento success, pero podría ser disparado un evento error en caso de que el objeto no pudiera ser leído por alguna circunstancia en particular. La función mostrarlista() toma el objeto devuelto por la propiedad result y almacena su valor en la variable resultado. El valor es el objeto devuelto desde el almacén de objetos, así que para acceder a su contenido, escribimos la variable que representa al objeto y el nombre de la propiedad (en este ejemplo, resultado.id). La variable resultado representa al objeto y id es una de las propiedades de éste.

15.2.6 Finalizar y probar el código

Del mismo modo que cualquiera de los códigos previos, este ejemplo debe ser finalizado agregando un detector o escucha para el evento load que ejecute la función iniciar() tan pronto como la aplicación sea cargada en la ventana del navegador:



Código 15-8

Iniciar la aplicación.

Hágalo usted mismo

Es momento de probar la aplicación en el navegador. Copie las declaraciones Javascript presentadas desde el Código 15-3 al Código 15-8 en el archivo llamado indexed.js y abra el documento HTML del Código 15-1. Usando el formulario que se mostrará en pantalla, inserte información acerca de las películas listadas al comienzo de este capítulo. Cada vez que una nueva película sea insertada, la información deberá ser mostrada en la caja de la derecha.

15.3 Listar datos

El método get() implementado en el Código 15-7 retorna un objeto cada vez (el último insertado). En el siguiente ejemplo vamos a usar un cursor para generar una lista que incluya todas las películas almacenadas en el almacén de objetos peliculas.

15.3.1 Cursores

Los cursores son una alternativa ofrecida por la API para obtener y navegar a través de un grupo de objetos encontrados en la base de datos. Un cursor obtiene una lista específica de objetos de un almacén de objetos e inicia un puntero que señala a un objeto de la lista a la vez. Para generar un cursor, la API proporciona el método openCursor(). Este método extrae información del almacén de objetos seleccionado y retorna un objeto IDBCursor que tiene sus propios métodos para manipular el cursor:

continue(indice): Este método mueve el puntero del cursor una posición y dispara nuevamente al evento success del cursor. Cuando el puntero alcanza el final de la lista, el evento success es disparado nuevamente pero retorna un objeto vacío. El cursor puede ser movido a una posición específica declarando un valor de índice dentro de los paréntesis del método.

delete(): Este método elimina el objeto que se encuentra en la posición actual del cursor.

update(valor): Este método es similar a put() pero modifica el valor del objeto en la posición actual del cursor.

El método openCursor() también tiene propiedades para especificar el tipo de objetos retornados y su orden. Los valores por defecto retornan todos los objetos disponibles en el almacén de objetos seleccionado, organizados en orden ascendente. Estudiaremos este tema más adelante.

```
  var cajadatos, bd
  function iniciar(){
    cajadatos = document.getElementById('cajadatos');
    var boton = document.getElementById('grabar');
    boton.addEventListener('click', agregarobjeto);

    var solicitud=indexedDB.open('mibase');
    solicitud.addEventListener('error', errores);
    solicitud.addEventListener('success', crear);
    solicitud.addEventListener('upgradeneeded', crearbd);

  }
  function errores(e){
  alert('Error: '+e.code+' '+e.message);
  }
  function crear(e){
    bd=e.target.result;
    mostrar();
  }

  function crearbd(e){
    var bd = e.target.result
    var almacen=bd.createObjectStore('peliculas',{keyPath:'id'});
    almacen.createIndex('BuscarFecha', 'fecha', {unique: false});
  }
  function agregarobjeto(){
  var clave=document.getElementById('clave').value;
  var titulo=document.getElementById('texto').value;
  var fecha=document.getElementById('fecha').value;

  var transaccion=bd.transaction(['peliculas'],"readwrite");
  var almacen=transaccion.objectStore('peliculas');
  var solicitud=almacen.add({id: clave, nombre: titulo, fecha:
              fecha});
  solicitud.addEventListener('success', mostrar);
  document.getElementById('clave').value='';
  document.getElementById('texto').value='';
  document.getElementById('fecha').value='';
}
function mostrar(){
  cajadatos.innerHTML = '';
  var transaccion = bd.transaction (['peliculas']);
  var almacen = transaccion.objectStore('peliculas');
  var nuevocursor = almacen.openCursor();
  nuevocursor.addEventListener('success', mostrarlista);

}
function mostrarlista(e){
  var cursor=e.target.result;
  if(cursor){
    cajadatos.innerHTML += '<div>' + cursor.value.id + ' - ' + cursor.
 value.nombre + ' - ' + cursor.value.fecha + '</div>';
    cursor.continue();
  }
}
addEventListener('load', iniciar);
```

Código 15-9

Lista de objetos.

El Código 15-9 muestra el código Javascript completo que necesitamos para este ejemplo. De las funciones usadas para configurar la base de datos, solo crear() presenta un pequeño cambio. Ahora al final de la función crear() es llamada la función mostrar() para mostrar la lista de objetos del almacén de objetos en pantalla, tan pronto como sea cargado el documento.

Añadimos las funciones mostrar() y mostrarlista(), y es en ellas donde trabajamos con cursores por primera vez.

La operación de leer información de la base de datos con un cursor también debe hacerse a través de una transacción. Por este motivo, lo primero que hacemos en la función mostrar() es generar una transacción del tipo readonly sobre el almacén de objetos peliculas.

Este almacén de objetos es seleccionado para la transacción y luego el cursor es abierto sobre el almacén de objetos mediante el método openCursor(). Si la operación es exitosa, un objeto es retornado con toda la información obtenida del almacén de objetos, un evento success es disparado desde este objeto y, finalmente, es ejecutada la función mostrarlista(). El objeto retornado por la operación ofrece varios atributos para leer la información. A continuación veremos de cuáles se trata.

key: Retorna el valor de la clave del objeto en la posición actual del cursor.

value: Retorna un objeto con los valores de las propiedades del objeto en la posición actual del cursor. Para leer estos valores debemos acceder a las propiedades desde la propiedad value usando una notación de punto, por ejemplo: value.fecha.

direction: Los objetos pueden ser leídos en orden ascendente o descendente (como veremos más adelante); este atributo retorna la condición actual en la cual son leídos.

count: Este atributo retorna un número aproximado de objetos en el cursor.

En la función mostrarlista() del Código 15-9, usamos la declaración if para controlar el contenido del cursor. Si ningún objeto es retornado o el puntero alcanza el final de la lista, entonces valor de la variable cursor será null (nulo) y el bucle terminará. Sin embargo, cuando el puntero apunta a un objeto válido, la información es mostrada en pantalla y el puntero es movido hasta la siguiente posición con continue().

Es importante mencionar que no usamos un bucle while porque el método continue dispara el evento success de nuevo, y la función completa es ejecutada nuevamente hasta que el cursor retorna null.

Hágalo usted mismo

El Código 15-9 reemplaza todos los códigos Javascript previos. Copie este nuevo código en el archivo. Abra la plantilla del Código 15-1 y, si aún no lo ha hecho, inserte todas las películas del listado encontrado al comienzo de este capítulo. Verá la lista completa de películas en la caja derecha de la pantalla en orden ascendente, de acuerdo al valor de la propiedad id.

15.3.2 Cambio de orden

Hay dos detalles que necesitamos modificar para obtener la lista que queremos. Todas las películas de nuestro ejemplo están listadas en orden ascendente y la propiedad usada para organizar los objetos es id. Esta propiedad es el atributo clave de las películas del almacén de objetos y, por tanto, es el índice usado por defecto, pero esta clase de valores no es la que a los usuarios les interesa conocer, al menos normalmente.

Considerando esta situación, en la función crearbd() añadimos otro índice a nuestro almacén. El nombre de este índice adicional fue declarado como BuscarFecha y la propiedad asignada al mismo es fecha. Este índice nos permitirá ordenar las películas de acuerdo al valor del año en el que fueron filmadas.

```
function mostrar(){
  cajadatos.innerHTML='';
  var transaccion=bd.transaction(['peliculas']);
  var almacen=transaccion.objectStore('peliculas');
  var indice=almacen.index('BuscarFecha');
  var nuevocursor=indice.openCursor(null, "prev");
  nuevocursor.addEventListener('success', mostrarlista);
}
```

Código 15-10

Orden descendiente por año.

La función del Código 15-10 reemplaza a la función mostrar() que usamos en el Código 15-9. Esta nueva función genera una transacción, luego asigna el índice BuscarFecha al almacén de objetos usado en la transacción y, finalmente, obtiene mediante openCursor() los objetos que tienen la propiedad correspondiente al índice (en este caso fecha).

Existen dos atributos que podemos usar para seleccionar y ordenar la información retornada por el cursor. El primer atributo declara el rango dentro del cual los objetos serán seleccionados y el segundo es una de las siguientes constantes:

next: El orden de los objetos retornados por openCursor será ascendente (es el valor por defecto).

nextunique: El orden de los objetos retornados será ascendente y los objetos duplicados serán ignorados (solo es retornado el primer objeto si varios comparten el mismo valor de índice).

prev: El orden de los objetos retornados será descendente.

prevunique: El orden de los objetos retornados será descendente y los

objetos duplicados serán ignorados (solo el primer objeto es retornado si varios comparten el mismo valor de índice).

Con el método openCursor() usado en la función mostrar() presentada en el Código 15-10, obtenemos los objetos en orden descendente y declaramos el atributo rango como null. Vamos a aprender a construir un rango y retornar solo los objetos deseados más adelante en este capítulo.

Hágalo usted mismo

Tome el Código 15-9 y reemplace la función show() con la nueva presentada en el Código 15-10. Esta nueva función lista las películas en la pantalla por año y en orden descendente (las más nuevas primero).

Debería otener el siguiente resultado:

id: tt1285016 nombre: La Red Social fecha: 2010

id: tt0111161 nombre: Cadena Perpetua fecha: 1994

id: tt0086567 nombre: Juegos de Guerra fecha: 1983

id: tt0068646 nombre: El Padrino fecha: 1972

15.4 Eliminar datos

Hemos aprendido a agregar, leer y listar datos. Es hora de estudiar la posibilidad de eliminar objetos de un almacén de objetos. Como mencionamos anteriormente, el método delete() provisto por la API recibe un valor y elimina el objeto cuya la clave corresponde a ese valor.

El código es sencillo; solo necesitamos crear un botón para cada objeto listado en pantalla y generar una transacción readwrite para poder realizar la operación de eliminación:

```
function mostrarlista(e){
  var cursor= e.target.result;
  if(cursor){
    cajadatos.innerHTML+='<div>'+cursor.value.id+' - '+cursor.value.
 nombre+' - '+cursor.value.fecha+' <input type="button"
              onclick="eliminar(\'' + cursor.value.id + '\')"
              value="Eliminar"></div>';

  cursor.continue();
 }
}
function eliminar(clave){
  if(confirm('¿Está Seguro?')){
   var transaccion=bd.transaction(['peliculas'], "readwrite");
   var almacen=transaccion.objectStore('peliculas');
   var solicitud=almacen.delete(clave);
   solicitud.addEventListener('success', mostrar, false);
  }
}
```

Código 15-11

Eliminar objetos.

Hágalo usted mismo

Tome el Código 15-9, reemplace la función mostrarlista() y agregue la función eliminar() del Código 15-11. Finalmente, abra el documento HTML del Código 15-1 para probar la aplicación. Podrá ver la lista de películas pero ahora cada línea incluye un botón para eliminar la película del almacén de objetos. Experimente agregando y eliminando películas.

El botón agregado para cada objeto en la función mostrarlista() del Código 15-11 contiene un atributo de evento. Cada vez que el usuario hace clic en uno de estos botones, la función eliminar() es ejecutada con el valor de la propiedad id como su atributo. Esta función genera primero una transacción readwrite y luego, mediante la clave recibida, procede a eliminar el correspondiente objeto del almacén de objetos peliculas.

Al final, si la operación fue exitosa, el evento success es disparado y la función mostrar() es ejecutada para actualizar la lista de películas en pantalla.

15.5 Buscar datos

Probablemente la operación más importante realizada en un sistema de base de datos es la búsqueda. El propósito de esta clase de sistemas es indexar la información almacenada para facilitar su posterior búsqueda. Como estudiamos anteriormente es este capítulo, el método get() es útil para obtener un objeto por vez cuando conocemos el valor de su clave, pero una operación de búsqueda usualmente es más compleja que esto.

Para obtener una lista específica de objetos desde el almacén de objetos, tenemos que usar un rango como argumento para el método openCursor(). La API incluye la interface IDBKeyRange con varios métodos y propiedades para declarar un rango y limitar los objetos retornados:

only(valor): Solo los objetos con la clave que concuerda con valor son retornados. Por ejemplo, si buscamos películas por año usando only(“1972”), la película El Padrino será la única devuelta desde la lista.

bound(bajo, alto, bajoAbierto, altoAbierto): Para realmente crear un rango, debemos contar con valores que indiquen el comienzo y el final de la lista, y además especificar si esos valores también serán incluidos en ella. El valor del atributo bajo indica el punto inicial de la lista y el atributo alto es el punto final. Los atributos bajoAbierto y altoAbierto son valores booleanos usados para declarar si los objetos que coinciden exactamente con los valores de los atributos bajo y alto serán ignorados. Por ejemplo, bound(“1972”, “2010”, false, true) retornará una lista de películas filmadas desde el año 1972 hasta el año 2010, pero no incluirá las realizadas específicamente en el 2010 debido a que el valor es true (verdadero) para el punto donde el rango termina y las películas de ese año no son incluidas.

lowerBound(valor, abierto): Este método crea un rango abierto que comienza en valor y acaba al final de la lista. Por ejemplo, lowerBound(“1983”, true) devuelve todas las películas hechas después de 1983, sin incluir las filmadas en ese año en particular.

upperBound(valor, abierto): Éste es el opuesto al anterior. Creará un rango abierto, con los objetos, desde el inicio de la lista hasta valor. Por ejemplo, upperBound(“1983”, false) retornará todas las películas hechas antes de 1983, incluyendo las realizadas ese mismo año.

Preparemos primero una nueva plantilla para presentar un formulario en pantalla con el que se pueda buscar películas:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>API IndexedDB</title>
  <link rel="stylesheet" href="indexed.css">
  <script src="indexed.js"></script>
</head>
<body>
  <section id="cajaformulario">
    <form name="formulario">
      <label for="año">Buscar Película por Año: </labe><br>
      <input type="text" name="fecha" id="fecha"><br>
       <input type="button" id="buscar">
    </form>
  </section>
  <section id="cajadatos">
   No hay información disponible
  </section>
</body>
</html>
```

Código 15-12

Formulario de búsqueda.

Este nuevo documento HTML provee un botón y un campo de texto donde ingresar el año para buscar películas de acuerdo a un rango especificado en el siguiente código:

```
var cajadatos, bd;
function iniciar(){
  cajadatos = document.getElementById('cajadatos');
  var boton = document.getElementById('buscar');
  boton.addEventListener('click', buscarobjetos);

  var solicitud=indexedDB.open('mibase');
  solicitud.addEventListener('error', errores);
  solicitud.addEventListener('success', crear);
  solicitud.addEventListener('upgradeneeded', crearbd);

}
function errores(e){
alert('Error: '+e.code+' '+e.message);
}
function crear(e){
  bd=e.target.result;
}

function crearbd(e){
  var bd = e.target.result
  var almacen=bd.createObjectStore('peliculas',{keyPath:'id'});
  almacen.createIndex('BuscarFecha', 'fecha', {unique: false});
}

function buscarobjetos(){
  cajadatos.innerHTML='';
  var buscar=document.getElementById('fecha').value;

  var transaccion=bd.transaction(['peliculas']);
  var almacen=transaccion.objectStore('peliculas');
  var indice=almacen.index('BuscarFecha');
  var rango=IDBKeyRange.only(buscar);

  var nuevocursor=indice.openCursor(rango);
  nuevocursor.addEventListener('success', mostrarlista);

}
function mostrarlista(e){
  var cursor=e.target.result;
  if(cursor){
   cajadatos.innerHTML += '<div>' + cursor.value.id + ' - '+ cursor.
          value.nombre + ' -    ' + cursor.value.fecha + '</div>';
    cursor.continue();
  }
}
window.addEventListener('load', iniciar);
```

Código 15-13

Buscando películas.

La función buscarobjetos() es la más importante del Código 15-13. En esta función generamos una transacción de solo lectura (readonly) para el almacén de objetos peliculas, seleccionamos el índice BuscarFecha para usar la propiedad fecha como índice, y creamos un rango que incluye los objetos cuya clave es igual a la de la variable fecha (el año insertado en el formulario). El método usado para construir el rango es only(), pero puede probar cualquiera de los métodos estudiados antes. Este rango es pasado luego como un atributo del método openCursor(). Si la operación es exitosa, la función mostrarlista() imprimirá en pantalla la lista de películas del año seleccionado.

El método only() retorna solo las películas que concuerdan exactamente con el valor de la variable buscar. Para probar otros métodos, puede usar sus propios valores para los atributos, por ejemplo, bound(buscar, “2011”, false, true).

El método openCursor() puede tomar dos posibles atributos al mismo tiempo. Por esta razón, una instrucción como openCursor(rango, “prev”) es válida y retornará los objetos en el rango especificado y en orden descendente (usando como referencia el mismo índice utilizado para el rango).

16 API File

16.1 Almacenamiento de archivos

Los archivos son unidades de información que los usuarios pueden fácilmente compartir con otras personas. Los usuarios no pueden compartir el valor de una variable pero pueden hacer copias de sus archivos y compartirlos mediante un DVD, discos duros portátiles, Internet, etc. Los archivos pueden almacenar grandes cantidades de datos y ser movidos, duplicados o transmitidos independientemente de la naturaleza de su contenido.

Los archivos fueron siempre una parte esencial de cada aplicación, pero hasta ahora no había forma posible de trabajar con ellos en la Web. Las opciones estaban limitadas a subir o descargar archivos ya existentes en servidores u ordenadores de usuarios. No existía la posibilidad de crear archivos, copiarlos o procesarlos en la web, hasta que llegó HTML5. La especificación de HTML5 fue desarrollada considerando cada aspecto de la construcción y la funcionalidad de las aplicaciones web. Desde el diseño hasta la estructura elemental de los datos, todo fue incluido. Y los archivos no podían ser ignorados, por supuesto. Por esta razón, la especificación incorpora la API File.

La API File comparte algunas características con las API de almacenamiento estudiadas en capítulos previos. Esta API posee una infraestructura de bajo nivel, aunque no tan compleja como IndexedDB y, al igual que otras, puede trabajar de forma síncrona o asíncrona. La parte síncrona fue desarrollada para ser usada con la API Web Workers (del mismo modo que IndexedDB y otras API), y la parte asíncrona está destinada a aplicaciones web convencionales. Estas características nos obligan nuevamente a cuidar cada aspecto del proceso, controlar si la operación fue exitosa o devolvió errores y, probablemente, adoptar (o desarrollar nosotros mismos) en el futuro algunas API más simples construidas sobre la misma.

API File es una vieja API que ha sido mejorada y expandida. Hoy en día está compuesta por tres especificaciones: API File, API File: Directories & System y API File: Writer, pero esta situación puede cambiar próximamente con la incorporación de nuevas especificaciones o incluso la unificación de algunas de las ya existentes. Básicamente, la API File nos permite interactuar con archivos locales y procesar su contenido en nuestra aplicación, la extensión la API File: Directories & System provee las herramientas para trabajar con un pequeño sistema de archivos creado específicamente para cada aplicación, y la extensión API File: Writer se usa para escribir contenido dentro de archivos que han sido creados o descargados por la aplicación.

16.2 Procesar archivos de usuario

Trabajar con archivos locales desde aplicaciones web puede ser peligroso. Los navegadores deben considerar medidas de seguridad antes de siquiera contemplar la posibilidad de permitir que las aplicaciones tengan acceso a los archivos del usuario. A este respecto, la API File provee solo dos métodos para cargar archivos desde una aplicación: la etiqueta <input> y la operación de arrastrar y soltar.

Importante

Esta API y sus extensiones no trabajan en este momento desde un servidor local, y solo Google Chrome y Firefox tienen implementaciones disponibles. Para ejecutar los códigos de este capítulo, deberá subir todos los archivos a su servidor.

En el Capítulo 13 aprendimos cómo usar la API Drag and Drop para arrastrar archivos desde una aplicación de escritorio y soltarlos dentro de la página web. La etiqueta <input>, cuando es usada con el tipo file, trabaja de forma similar a la API Drag and Drop. Ambas técnicas transmiten archivos a través de la propiedad files. Del mismo modo que lo hicimos en ejemplos previos, lo único que debemos hacer es explorar el valor de esta propiedad para obtener los archivos que han sido seleccionados o arrastrados.

16.2.1 Plantilla

En esta primera parte del capítulo vamos a usar la etiqueta <input> para seleccionar archivos, pero usted puede, si lo desea, aprovechar la información del Capítulo 13 para integrar estos códigos con la API Drag and Drop.

```
<!DOCTYPE html>
<html lang="es">
<head>
 <title>File API</title>
 <link rel="stylesheet" href="file.css">
 <script src="file.js"></script>
</head>
<body>
 <section id="cajaformulario">
 <form name="formulario">
   <label for="archivos">File: </label><br>
   <input type="file" name="archivos" id="archivos">
 </form>
 </section>
 <section id="cajadatos">
   No se seleccionaron archivos
 </section>
</body>
</html>
```

Código 16-1

Plantilla para trabajar con los archivos del usuario.

El archivo CSS incluye estilos para esta plantilla y otros que vamos a usar más adelante:

```
#cajaformulario{
 float: left;
 padding: 20px;
 border: 1px solid #999999;
}
#cajadatos{
 float: left;
 width: 500px;
 margin-left: 20px;
 padding: 20px;
 border: 1px solid #999999;

}
  .directorio{
 color: #0000FF;
 font-weight: bold;
 cursor: pointer;
}
```

Código 16-2

Estilos para el formulario y cajadatos.

16.2.2 Leer archivos

La API File proporciona un constructor para obtener el objeto FileReader (lector de archivos). Este objeto incluye propiedades, métodos y eventos para leer los archivos del usuario desde su ordenador.

FileReader(): Este constructor devuelve un objeto FileReader. El objeto debe ser creado antes de leer los archivos. El proceso es asíncrono y el resultado es informado a través de eventos.

Para obtener el contenido del archivo, el objeto FileReader incluye los siguientes métodos:

readAsText(archivo, codificación): Para procesar el contenido como texto podemos usar este método. El contenido es retornado codificado como texto UTF-8 a menos que el atributo codificación sea especificado con un valor diferente. Este método intentará interpretar cada byte o una secuencia de múltiples bytes como caracteres de texto. Un evento load es disparado desde el objeto FileReader cuando el archivo es cargado.

readAsBinaryString(archivo): La información es leída por este método como una sucesión de enteros en el rango de 0 a 255. Este método nos asegura que cada byte es leído como es, sin ninguna interpretación. Es útil para procesar contenido binario como imágenes o videos.

readAsDataURL(archivo): Este método genera una cadena del tipo data:url

codificada en base64 que representa los datos del archivo.

readAsArrayBuffer(archivo): Este método retorna los datos del archivo como datos del tipo ArrayBuffer.

```
var cajadatos;
function iniciar(){
 cajadatos=document.getElementById('cajadatos');
 var archivos=document.getElementById('archivos');
 archivos.addEventListener('change', procesar);
}
function procesar(e){
 var archivos=e.target.files;
 var archivo=archivos[0];
 var lector=new FileReader();
 lector.addEventListener('load', mostrar);
 lector.readAsText(archivo);
}

function mostrar(e){
 var resultado=e.target.result;
 cajadatos.innerHTML=resultado;
}
addEventListener('load', iniciar);
```

Código 16-3

Leer un archivo de texto.

El campo de entrada llamado archivos del documento HTML del Código 16-1 permite al usuario seleccionar un archivo para ser procesado. Para detectar esta acción, en la función iniciar() del Código 16-3 agregamos un detector para el evento change. Gracias a este detector, se ejecuta la función procesar()cada vez que algo cambia en el elemento <input>.

Importante

Para aprender más acerca del atributo multiple, lea nuevamente el Capítulo 5, Código 5-22. También puede encontrar un ejemplo de cómo trabajar con múltiples archivos en el Capítulo 13, Código 13-10.

La propiedad files enviada por el elemento <input> (y también por la API Drag and Drop) es una matriz que contiene todos los archivos seleccionados. Cuando el atributo multiple no está presente en la etiqueta <input> no es posible seleccionar múltiples archivos, por lo que el único elemento disponible será el primero de la matriz. Al comienzo de la función procesar() tomamos el contenido de la propiedad files, lo almacenamos en la variable archivos y luego seleccionamos el primer archivo con la instrucción var archivo=archivos[0].

Lo primero que debemos hacer para comenzar a procesar el archivo es obtener un objeto FileReader usando el constructor FileReader(). En la función procesar() del Código 16-3 llamamos lector a este objeto. En el siguiente paso, registramos un detector para el evento load con el objetivo de detectar cuándo se carga el archivo y podemos comenzar a procesarlo. Finalmente, el método llamado readAsText() lee el archivo y retorna su contenido en formato texto. Cuando el método readAsText() finaliza la lectura del archivo, el evento load es disparado y se llama a la función mostrar(). Esta función toma el contenido del archivo desde la propiedad result del objeto lector y lo muestra en pantalla.

Este código, por supuesto, espera recibir archivos de texto, pero el método readAsText() toma lo que le enviamos y lo interpreta como texto, incluyendo archivos con contenido binario (por ejemplo, imágenes). Si carga archivos con contenidos que no son de texto, verá aparecer caracteres extraños en la pantalla.

Hágalo usted mismo

Cree archivos con el contenido de los Códigos 16-1, 16-2 y 16-3. Recuerde que los nombres para los archivos CSS y Javascript fueron declarados en el documento HTML como file.css y file.js respectivamente. Abra la plantilla en el navegador y use el formulario para seleccionar un archivo en su ordenador. Intente usar archivos de texto así como imágenes para ver cómo se presenta el contenido en pantalla.

16.2.3 Propiedades de archivos

En una aplicación real, es necesario proporcionar información como el nombre del archivo, su tamaño o tipo, para informar al usuario sobre los archivos que están siendo procesados o incluso controlar cuáles son o no son admitidos. El objeto File enviado por la etiqueta <input> incluye varias propiedades para acceder a esta información:

name: Esta propiedad retorna el nombre completo del archivo (nombre y extensión).

size: Esta propiedad retorna el tamaño del archivo en bytes.

type: Esta propiedad retorna el tipo de archivo, especificado en tipos MIME.

```
var cajadatos;
function iniciar(){
 cajadatos=document.getElementById('cajadatos');
 var archivos=document.getElementById('archivos');
 archivos.addEventListener('change', procesar);
}
function procesar(e){
 var archivos=e.target.files;
 cajadatos.innerHTML = '';
 var archivo=archivos[0];
 if(!archivo.type.match(/image.*/i)){
 alert('seleccione una imagen');
 }else{
 cajadatos.innerHTML+='Nombre: '+archivo.name+'<br>';
 cajadatos.innerHTML+='Tamaño: '+archivo.size+' bytes<br>';

 var lector=new FileReader();
 lector.addEventListener('load', mostrar);
 lector.readAsURL(archivo);
 }
}
function mostrar(e){
 var resultado=e.target.result;
 cajadatos.innerHTML+='<img src="'+resultado+'">';
}
addEventListener('load', iniciar);
```

Código 16-4

Cargar imágenes.

El ejemplo del Código 16-4 es similar al anterior excepto que esta vez usamos el método readAsDataURL() para leer el archivo. Este método retorna el contenido del archivo en el formato data:url que puede ser usado luego como fuente de un elemento <img> para mostrar la imagen seleccionada en la pantalla.

Cuando necesitamos procesar una clase particular de archivo, lo primero que debemos hacer es leer la propiedad type del archivo. En la función procesar() del Código 16-4 controlamos este valor aprovechando el viejo método match(). Si el archivo no es una imagen, mostramos un mensaje de error con alert(). Si, en cambio, el archivo es efectivamente una imagen, el nombre y tamaño del archivo son mostrados en pantalla y el archivo es abierto.

Apartando el uso de readAsDataURL(), el proceso de apertura es exactamente el mismo. Es creado el objeto FileReader, se añade un controlador al evento load y el archivo es cargado. Una vez que el proceso termina, la función mostrar() usa el contenido de la propiedad result como fuente del elemento <img> y la imagen seleccionada es mostrada en la pantalla.

Conceptos básicos

Para construir el filtro usamos expresiones regulares y el conocido método

Javascript match(). Este método busca cadenas de texto que concuerden con la expresión regular, devolviendo una matriz con todas las coincidencias o el valor null en caso de no encontrar ninguna. El tipo MIME para imágenes es algo así: image/jpeg para imágenes en formato JPG, o image/gif para imágenes en formato gif, por lo que la expresión /image.*/i aceptará cualquier formato de imagen, pero otro tipo de archivos. Para más información sobre expresiones regulares o tipos MIME, visite nuestro sitio web y siga los enlaces correspondientes a este capítulo.

16.2.4 Blobs

Además de archivos data:url, la API trabaja con otra clase de fuente de datos llamados blobs. Un blob es un objeto que representa datos “en crudo”. Fue creado con el propósito de superar las limitaciones de Javascript para trabajar con datos binarios. Un blob es normalmente generado a partir de un archivo, aunque no necesariamente. Es una buena alternativa para trabajar con datos sin cargar archivos enteros en memoria, y hace posible procesar información binaria en pequeñas piezas.

Un blob tiene propósitos múltiples, pero está enfocado a ofrecer una mejor manera de procesar grandes cantidades de datos crudos o archivos grandes. Para generar blobs desde otros blobs o archivos, la API ofrece el método slice():

slice(comienzo, largo, tipo): Este método retorna un nuevo blob generado desde otro blob o un archivo. El primer atributo indica el punto de inicio, el segundo el largo del nuevo blob, y el último es un atributo opcional para especificar el tipo de datos usados.

```
var cajadatos;
function iniciar(){
 cajadatos=document.getElementById('cajadatos');
 var archivos=document.getElementById('archivos');
 archivos.addEventListener('change', procesar);
}
function procesar(e){
 cajadatos.innerHTML='';
 var archivos=e.target.files;
 var archivo=archivos[0];
 var lector=new FileReader();
 lector.addEventListener('load', function(e){mostrar(e, archivo)});
 var blob=archivo.slice(0,1000);
 lector.readAsBinaryString(blob);
}
function mostrar(e, archivo){
 var resultado=e.target.result;
 cajadatos.innerHTML='Nombre: '+archivo.name+'<br>';
 cajadatos.innerHTML+='Tipo: '+archivo.type+'<br>';
 cajadatos.innerHTML+='Tamaño: '+archivo.size+' bytes<br>';
 cajadatos.innerHTML+='Tamaño Blob: '+resultado.length+' bytes<br>';
 cajadatos.innerHTML+='Blob: '+resultado;
}
addEventListener('load', iniciar);
```

Código 16-5

Trabajar con blobs.

En el Código 16-5 hicimos exactamente lo mismo que veníamos haciendo hasta el momento, pero esta vez en lugar de leer el archivo completo creamos un blob con el método slice(). El blob tiene 1000 bytes y comienza desde el byte 0 del archivo. Si el archivo cargado tiene menos de 1000 bytes, el blob será del mismo tamaño del archivo (desde el comienzo hasta el EOF, o End Of File).

Para mostrar la información obtenida por este proceso, usamos una función anónima que detecta el evento load. Esta función llama a la función mostrar() con una referencia al objeto archivo como atributo. La referencia es recibida por la función mostrar, y los valores de las propiedades de archivo son mostradas en la pantalla.

Las ventajas ofrecidas por los blobs son incontables. Podemos crear un bucle para generar varios blobs a partir de un archivo, por ejemplo, y luego procesar esta información por partes, crear programas asíncronos para subir archivos al servidor o aplicaciones para procesar imágenes, entre otras cosas. Los blobs ofrecen nuevas posibilidades a los códigos Javascript.

16.2.5 Eventos

El tiempo que toma cargar un archivo en la memoria depende del tamaño de aquél. Para archivos pequeños, el proceso parece ser instantáneo, pero los archivos grandes pueden tardar varios segundos en cargarse. Además del evento load ya estudiado, la API ofrece varios eventos especiales para informar sobre cada instancia del proceso:

loadstart: Este evento es disparado desde el objeto FileReader cuando la carga comienza.

progress: Este evento es disparado periódicamente mientras el archivo o blob está siendo leído.

abort: Este evento es disparado si el proceso es abortado.

error: Este evento es disparado cuando la carga ha fallado.

loadend: Este evento es similar a load, pero es disparado en caso de éxito o fracaso.

```
var cajadatos
function iniciar(){
 cajadatos=document.getElementById('cajadatos');
 var archivos=document.getElementById('archivos');
 archivos.addEventListener('change', procesar);
}
 function procesar(e){
 cajadatos.innerHTML='';
 var archivos=e.target.files;
 var archivo=archivos[0];
 var lector=new FileReader();
 lector.addEventListener('loadstart', comenzar);
 lector.addEventListener('progress', estado);
 lector.addEventListener('loadend', function(){ mostrar(archivo);
 });
 lector.readAsBinaryString(archivo);
}
function comenzar(e){
 cajadatos.innerHTML='<progress value="0" max="100">0%</progress>';
}
function estado(e){
 var por=parseInt(e.loaded/e.total*100);
 cajadatos.innerHTML='<progress value="'+por+'" max="100">'+por+'%</
 progress>';
}
function mostrar(archivo){
 cajadatos.innerHTML='Nombre: '+archivo.name+'<br>';
 cajadatos.innerHTML+='Tipo: '+archivo.type+'<br>';
 cajadatos.innerHTML+='Tamaño: '+archivo.size+' bytes<br>';
}
addEventListener('load', iniciar

);
```

Código 16-6

Usar eventos para controlar el proceso de lectura.

Con el Código 16-6 creamos una aplicación que carga un archivo y muestra el progreso de la operación a través de una barra de progreso. Tres controladores de eventos para el objeto FileReader controlan el proceso de lectura y dos funciones responden a estos eventos: comenzar() y estado(). La función comenzar() muestra la barra de progreso con el valor 0% en pantalla. Esta barra de progreso podría usar cualquier valor o rango, pero decidimos usar porcentajes para que sea más comprensible para el usuario. En la función estado(), este porcentaje es calculado a partir de las propiedades loaded y total retornadas por el evento progress. La barra de progreso se actualiza en la pantalla cada vez que el evento progress es disparado.

Hágalo usted mismo

Usando la plantilla del Código 16-1 y las instrucciones Javascript del Código 16-6, pruebe cargar un archivo extenso (puede intentar con un vídeo, por ejemplo) para ver la barra de progreso en funcionamiento. Si el navegador no reconoce el elemento <progress>, el contenido de este elemento es mostrado en su lugar.

Importante

En nuestro ejemplo utilizamos innerHTML para agregar un nuevo elemento <progress> al documento. Esta no es una práctica recomendada pero es útil y conveniente en este caso por razones didácticas. Normalmente los elementos son agregados al documento usando el método Javascript createElement() junto con appendChild(), tal como explicamos en el Capítulo 4.

16.3 Crear archivos

La parte principal de la API File es útil para cargar y procesar archivos ubicados en el ordenador del usuario, pero trabaja solo con archivos que ya existen en el disco duro. No contempla la posibilidad de crear nuevos archivos o directorios. Una extensión de esta API llamada API File: Directories & System se hace cargo de estas otras acciones situación. La API reserva un espacio específico en el disco duro, un espacio de almacenamiento especial en el cual la aplicación web podrá crear y procesar archivos y directorios exactamente como lo haría una aplicación de escritorio. El espacio es único y solo accesible por la aplicación que lo creó.

16.3.1 Documento HTML

Para probar esta parte de la API File vamos a necesitar un nuevo formulario con un campo de texto, así como un botón para crear y procesar tanto archivos como directorios:

```
<!DOCTYPE html>
<html lang="es">
<head>
 <title>API File</title>
 <link rel="stylesheet" href="file.css">
 <script src="file.js"></script>
</head>

<body>
 <section id="cajaformulario">
  <form name="formulario">
   <label for="entrada">Nombre:</label><br>
   <input type="text" name="entrada" id="entrada">
   <input type="button" id="boton" value="Guardar">
  </form>
 </section>
 <section id="cajadatos">
No hay entradas disponibles
 </section>
</body>
</html>
```

Código 16-7

Nueva plantilla para API File: Directories & System.

Hágalo usted mismo

El documento HTML genera un nuevo formulario pero preserva la misma estructura y estilos CSS. Para probar los siguientes ejemplos, solo necesita reemplazar el código HTML anterior por el del Código 16-7 y copiar los códigos Javascript dentro del archivo file.js.

16.3.2 El disco duro

El espacio reservado para la aplicación es como un espacio aislado, una pequeña unidad de disco duro con su propio directorio raíz y configuración. Para comenzar a trabajar con esta unidad virtual, primero tenemos que solicitar que un sistema de archivos sea inicializado para nuestra aplicación.

requestFileSystem(tipo, tamaño, función éxito, función error): Este método crea un sistema de archivos del tamaño y tipo especificados por sus atributos. El valor del atributo tipo puede ser TEMPORARY (temporal) o PERSISTENT (permanente) de acuerdo al tiempo que deseamos que los archivos sean preservados. El atributo tamaño determina el espacio total que será reservado en el disco duro para este sistema de archivos en bytes. En caso de error o éxito, el método llama a las correspondientes funciones (atributos success y error).

Importante

Actualmente Google Chrome es el único navegador con una implementación funcional de esta parte de la API File. Debido a que la implementación es experimental, tenemos que reemplazar el método requestFileSystem() por el específico de Chrome webkitRequestFileSystem(). Usando este método, podrá probar en su navegador los códigos de los ejemplos de este apartado.

El método requestFileSystem() retorna un objeto FileSystem con dos

propiedades:

root: El valor de esta propiedad es una referencia al directorio raíz del sistema de archivos. Retorna un objeto DirectoryEntry (entrada de directorio) y cuenta con importantes métodos para trabajar con archivos y directorios, como veremos más adelante.

name: Esta propiedad retorna información acerca del sistema de archivos, como el nombre asignado por el navegador y su condición.

```
var cajadatos, dd;
function iniciar(){
 cajadatos=document.getElementById('cajadatos');
 var boton=document.getElementById('boton');
 boton.addEventListener('click', crear);
 webkitRequestFileSystem(TEMPORARY, 5*1024*1024, creardd, errores);
}
function creardd(fs) {
 dd=fs.root;
}
function crear(){
 var nombre=document.getElementById('entrada').value;
 if(nombre!=''){
 dd.getFile(nombre, {create: true, exclusive: false}, mostrar,

 errores);
 }
}
function mostrar(entrada){
 document.getElementById('entrada').value='';
 cajadatos.innerHTML='¡Entrada Creada!<br>';
 cajadatos.innerHTML+='Nombre: '+entrada.name+'<br>';
 cajadatos.innerHTML+='Ruta: '+entrada.fullPath+'<br>';
 cajadatos.innerHTML+='Sistema: '+entrada.filesystem.name;
}
function errores(e){
 alert('Error: '+e.code);
}
addEventListener('load', iniciar);
```

Código 16-8

Crear un sistema de archivos propio.

Usando el documento HTML del Código 16-7 y las declaraciones Javascript del Código 16-8, obtenemos nuestra primera aplicación para trabajar con nuevos archivos en el ordenador del usuario. El código llama al método requestFileSystem() para crear el sistema de archivos (u obtener una referencia de éste si el sistema ya existe).

Si ésta es la primera visita, el sistema de archivos se crea como permanente, con un tamaño de 5 MB (5*1024*1024). En caso de que esta última operación sea un éxito, es ejecutada la función creardd() y el proceso de inicialización continúa. Controlarmos los errores mediante la función errores(), del mismo modo que lo hicimos para otras APIs.

Cuando se abre o se crea el sistema de archivos, la función creardd() recibe un objeto FileSystem y graba el valor de la propiedad root en la variable dd para poder hacer referencia al directorio raíz más adelante.

Importante

El almacenamiento permanente exige una solicitud de espacio al navegador. Esta solicitud se realiza mediante la API Quota Management. Al momento de escribir este manual, la especificación de esta API está aún en proceso de desarrollo y

todavía no hay una implementación disponible para los navegadores. Por eso, en este ejemplo y en los siguientes, declaramos el atributo TEMPORARY para establecer un sistema de archivos temporal. Para confirmar el estado de la especificación de la API Quota Management u obtener información adicional sobre este asunto, visite nuestra página web y siga los vínculos disponibles para este capítulo.

16.3.3 Crear archivos

El proceso de iniciación del sistema de archivos está listo. El resto de las funciones del Código 16-8 crean un nuevo archivo y muestran los datos de la entrada en la pantalla. Al pulsar el botón Guardar del formulario, es llamada la función crear(), que asigna el texto insertado a la variable nombre, en el elemento <input>, y crea un archivo con ese nombre usando el método getFile(). Este último método es parte de la interfaz DirectoryEntry incluida en la API. La interfaz proporciona un total de cuatro métodos para crear y manejar archivos y directorios:

getFile(ruta, opciones, éxito, error): Este método crea o abre un archivo. El atributo ruta debe incluir el nombre del archivo y la ruta donde el archivo está localizado (desde la raíz de nuestro sistema de archivos). Hay dos banderas que podemos usar para configurar el comportamiento de este método: create y exclusive. Ambas reciben valores booleanos. La bandera create (crear) indica si el archivo será creado o no (en caso de que no exista, por supuesto), y el marcador exclusive (exclusivo), cuando es declarada como true, fuerza al método getFile() a devolver un error si intentamos crear un archivo que ya existe. Este método también recibe dos funciones para responder en caso de éxito o error.

getDirectory(ruta, opciones, éxito, error): Este método tiene exactamente las mismas características que el anterior pero es exclusivo para directorios (carpetas).

createReader(): Este método retorna un objeto DirectoryReader para leer entradas desde el directorio indicado.

removeRecursively(): Éste es un método específico para eliminar directorios y todo su contenido.

En el Código 16-8, el método getFile() usa el valor de la variable nombre para crear u obtener el archivo. El archivo será creado si no existe (create: true) o será leído en caso contrario (exclusive: false). La función crear() también controla el valor de la variable nombre antes de ejecutar getFile().

El método getFile() usa dos funciones, mostrar() y errores(), para responder al éxito o fracaso de la operación. La función mostrar() recibe un objeto Entry (entrada) y muestra el valor de sus propiedades en la pantalla. Este tipo de objetos tiene varios métodos y propiedades asociadas que estudiaremos más adelante. Por ahora hemos aprovechado solo las propiedades name, fullPath y filesystem.

16.3.4 Crear directorios

El método getFile() (específico para archivos) y el método getDirectory() (específico para directorios) son exactamente iguales. Para crear un directorio usando el documento del Código 16-7, solo hay que reemplazar el método getFile() con el método getDirectory(), como se muestra en el siguiente código:

```
function crear(){

 var nombre=document.getElementById('entrada').value;

 if(nombre!=''){

 dd.getDirectory(nombre, {create: true, exclusive: false}, mostrar,

                                errores);

 }

}
```

Código 16-9

Usar getDirectory() para crear un directorio.

Observe que ambos métodos son parte del objeto DirectoryEntry llamado root, que estamos representando con la variable dd, por lo que siempre deberemos usar esta variable para llamar a los métodos así como crear archivos y directorios en el sistema de archivos de nuestra aplicación.

Hágalo usted mismo

Use la función del Código 16-9 para reemplazar la función crear() del Código 16-8 y así crear directorios en lugar de archivos. Suba los archivos a su servidor, abra el documento HTML del Código 16-7 en su navegador y cree un directorio usando el formulario en la pantalla.

16.3.5 Listar archivos

Como mencionamos antes, el método createReader() nos permite acceder a una lista de entradas (archivos y directorios) en una ruta específica. Este método retorna un objeto DirectoryReader que contiene el método readEntries() para leer las entradas obtenidas:

readEntries(función éxito, función error): Este método lee el siguiente bloque de entradas desde el directorio seleccionado. Cada vez que es llamado el método, la función utilizada para procesar las operaciones exitosas retorna un objeto con la lista de entradas encontrada, o el valor null si no se encontró ninguna.

El método readEntries() lee la lista de entradas por bloques. Como consecuencia, no existe garantía alguna de que todas las entradas serán retornadas en una sola llamada. Tendremos que llamar al método tantas veces como sea necesario hasta que el objeto retornado sea un objeto vacío.

Además, deberemos hacer otra consideración antes de escribir nuestro próximo código. El método createReader() retorna un objeto DirectoryReader para un directorio específico. Para obtener los archivos que queremos, primero tenemos que obtener el correspondiente objeto Entry del directorio o archivo que queremos leer:

```
var cajadatos, dd, ruta;
function iniciar(){
 cajadatos=document.getElementById('cajadatos');
 var boton=document.getElementById('boton');
 boton.addEventListener('click', crear);

 webkitRequestFileSystem(TEMPORARY, 5*1024*1024, creardd, errores);
}
function creardd(sistema) {
 dd=sistema.root;
 ruta='';
 mostrar();

}
function errores(e){
 alert('Error: '+e.code);
}
function crear(){
 var nombre=document.getElementById('entrada').value;
 if(nombre!=''){
 nombre=ruta+nombre;
 dd.getFile(nombre, {create: true, exclusive: false}, mostrar,
 errores);
 }
}
function mostrar(){
 document.getElementById('entrada').value='';
 cajadatos.innerHTML='';
 dd.getDirectory(ruta,null,leerdir,errores);
}
function leerdir(dir){
 var lector=dir.createReader();
 var leer=function(){
  lector.readEntries(function(archivos){
   if(archivos.length){
    listar(archivos);
    leer();
  }
  }, errores);
 }
 leer();
}
function listar(archivos){
 for(var i=0; i<archivos.length; i++) {
  if(archivos[i].isFile) {
   cajadatos.innerHTML+=archivos[i].name+'<br>';
 }else if(archivos[i].isDirectory){
   cajadatos.innerHTML+='<span onclick="cambiardir(\''+archivos[i].
 name+'\')" class="directorio">+'+archivos[i].name+'</span><br>';
 }
 }
}
function cambiardir(nuevaruta){
 ruta=ruta+nuevaruta+'/';
 mostrar();
}

addEventListener('load', iniciar);
```

Código 16-10

Sistema de archivos completo.

Este código no reemplazará al Explorador de archivos de Windows, pero al menos proporcionará toda la información que necesitamos para entender cómo construir un sistema de archivos útil y funcional para la web. Vamos a analizarlo paso a paso.

La función iniciar() funciona igual que en códigos previos: inicia o crea el sistema de archivos y llama a la función creardd() si tiene éxito. Además de declarar la variable dd para poder hacer referencia al directorio raíz de nuestro disco duro virtual, la función creardd() también inicializa la variable ruta con una cadena de texto vacía (que representa el directorio raíz) y llama a la función mostrar() para mostrar la lista de entradas en pantalla tan pronto como se cargue la aplicación.

La variable ruta será usada en el resto de la aplicación para almacenar el valor de la ruta actual dentro del sistema de archivos en el que el usuario está trabajando. Para entender su importancia, observe los cambios realizados en la función crear() en el Código 16-10 que estamos analizando para usar este valor. Ahora, cada vez que es enviado desde el formulario un nuevo nombre de archivo, la ruta se agrega al nombre y el archivo se crea en el directorio actual.

Como ya explicamos, para mostrar la lista de entradas, debemos primero abrir el directorio que debe ser leído. Mediante el método getDirectory() de la función mostrar(), se abre el directorio actual de acuerdo a la variable ruta y se envía una referencia a este directorio a la función leerdir() si la operación es exitosa. Esta

función guarda la referencia en la variable dir, crea un nuevo objeto DirectoryReader para el directorio actual y obtiene la lista de entradas con el método readEntries().

En leerdir() son usadas funciones anónimas para mantener el código organizado y no recargar el entorno global. En primer lugar, createReader() crea un objeto DirectoryReader para el directorio representado por dir. Luego, es creada dinámicamente una nueva función llamada leer()para leer las entradas usando el método readEntries() y asignando una función anónima a la variable read. Este método lee las entradas por bloques, lo que significa que debemos llamarlo varias veces para asegurarnos de que todas las entradas disponibles en el directorio son leídas. La función leer() nos ayuda a lograr este propósito. El proceso es el siguiente: al final de la función leerdir(), es llamada la función leer() por primera vez.

Dentro de la función leer() llamamos al método readEntries(), que usa otra función anónima en caso de éxito para recibir el objeto files y procesar su contenido. Si este objeto no está vacío, es llamada la función listar() para mostrar en pantalla las entradas leídas, y la función leer() es ejecutada nuevamente para obtener el siguiente bloque de entradas. Esta función se llamará a sí misma una y otra vez hasta que no sea retornada ninguna entrada.

La función listar(), llamada por read(), se encarga de imprimir la lista de entradas (archivos y directorios) en pantalla. Toma el objeto files y comprueba las características de cada entrada usando dos propiedades importantes de la interfaz Entry, que son isFile (es archivo) e isDirectory (es directorio). Como sus nombres indican, estas propiedades contienen valores booleanos que señalan si la entrada es un archivo o un directorio. Luego de que esta condición es controlada, se usa la propiedad name para mostrar la información en la pantalla.

Existe una diferencia en cómo nuestra aplicación mostrará un archivo o un directorio en la pantalla. Cuando una entrada es detectada como directorio, es

mostrada a través de un elemento con un controlador de eventos onclick que llamará a la función cambiardir() si el usuario hace clic sobre el mismo. El propósito de esta función es declarar la nueva ruta actual para apuntar al directorio seleccionado. Recibe el nombre del directorio, agrega el directorio al valor de la variable ruta y llama a la función mostrar() para actualizar la información en pantalla.

Esta característica permite abrir directorios y ver su contenido con solo un clic del ratón, exactamente como lo haría un explorador de archivos común y corriente.

Este ejemplo no contempla la posibilidad de subir en la estructura para ver el contenido de directorios padre. Para hacerlo, debemos aprovechar otro método provisto por la interfaz Entry:

getParent(función éxito, función error): Este método retorna un objeto Entry del directorio que contiene la entrada seleccionada. Una vez que obtenemos el objeto Entry podemos leer sus propiedades para obtener toda la información acerca del directorio padre de esa entrada en particular.

Trabajar con el método getParent() es simple: supongamos que ha sido creada una estructura de directorios como fotos/misvacaciones y el usuario está listando el contenido de misvacaciones en este momento. Para regresar al directorio fotos, podríamos incluir un enlace en el documento HTML con un atributo de evento onclick que llame a la función encargada de modificar la ruta actual para apuntar a esta nueva dirección (el directorio fotos). La función llamada por este evento de tributo podría ser similar a la siguiente:

```
function volver(){
 dd.getDirectory(ruta,null,function(dir){
 dir.getParent(function(padre){
 ruta=padre.fullPath;
 mostrar();
 }, errores);
 },errores);
}
```

La función volver() del Código 16-11 cambia el valor de la variable ruta para apuntar al directorio padre del directorio actual. Lo primero que hacemos es obtener una referencia del directorio actual usando el método getDirectory(). Si la operación es exitosa, una función anónima es ejecutada. En esta función, el método getParent() es usado para encontrar el directorio padre del directorio al que hace referencia dir (el directorio actual). Si esta operación es exitosa, otra función anónima es ejecutada para recibir el objeto padre y declarar el valor de la ruta actual igual al valor de la propiedad fullPath (esta propiedad contiene la ruta completa hacia el directorio padre). La función mostrar() es llamada al final del proceso para actualizar la información en pantalla, mostrando las entradas ubicadas en la nueva ruta.

Por supuesto, esta aplicación puede ser extremadamente enriquecida y mejorada, pero eso es algo que dejamos en sus manos.

Hágalo usted mismo

Agregue la función del Código 16-11 al Código 16-10 y cree un enlace en el documento HTML para llamar a esta función (por ejemplo, volver).

16.3.6 Manejar archivos

Ya mencionamos que la interfaz Entry incluye un grupo de propiedades y métodos para obtener información y operar archivos. Muchas de las propiedades disponibles ya fueron usadas en ejemplos anteriores. Ya aprovechamos las propiedades isFile e isDirectory para comprobar la clase de entrada, y también usamos los valores de name, fullPath y filesystem para mostrar información sobre la entrada en pantalla. El método getParent(), estudiado en el último código, es también parte de esta interfaz. Sin embargo, existen todavía algunos métodos más que son útiles para realizar operaciones comunes sobre archivos y directorios. Usando estos métodos podremos mover, copiar y eliminar entradas exactamente como en cualquier aplicación de escritorio:

moveTo(directorio, nombre, función éxito, función error): Este método mueve una entrada a una ubicación diferente en el sistema de archivos. Si se proporciona el atributo nombre, el nombre de la entrada será cambiado a este valor.

copyTo(directorio, nombre, función éxito, función error): Este método genera una copia de una entrada en otra ubicación dentro del sistema de archivos. Si se proporciona el atributo nombre, el nombre de la nueva entrada será cambiado a este valor.

remove(): Este método elimina un archivo o un directorio vacío (para eliminar un directorio con contenido, debemos usar el método removeRecursively() presentado anteriormente).

Necesitaremos una nueva plantilla para probar estos métodos. Para simplificar los códigos, vamos a crear un formulario con solo dos campos, uno para el origen y otro para el destino de cada operación:

```
<!DOCTYPE html>
<html lang="es">
<head>
 <title>File API</title>
 <link rel="stylesheet" href="file.css">
 <script src="file.js"></script>
</head>
<body>
 <section id="cajaformulario">
  <form name="formulario">
   <label for="origen">Origen: </label>
   <input type="text" name="origen" id="origen" required><br>
   <label for="destino">Destino:</label>
   <input type="text" name="destino" id="destino" required>
   <br><input type="button" id="boton" value="Aceptar">
  </form>
 </section>
 <section id="cajadatos"></section>
</body>
</html>
```

Código 16-12

Nueva plantilla para operar con archivos

16.3.7 Mover archivos

El método moveTo() requiere un objeto Entry para el archivo y otro para el directorio en donde el archivo será movido. Por lo tanto, primero tenemos que crear una referencia al archivo que vamos a mover usando getFile(), luego obtenemos la referencia del directorio destino con getDirectory(), y finalmente aplicamos moveTo() con esta información:

```
var cajadatos, dd, ruta;
function iniciar(){
 cajadatos=document.getElementById('cajadatos');
 var boton=document.getElementById('boton');
 boton.addEventListener('click', modificar);

 webkitRequestFileSystem(TEMPORARY, 5*1024*1024, creardd, errores);
}
function creardd(sistema){
 dd=sistema.root;
 ruta='';
 mostrar();
} function errores(e){
 alert('Error: '+e.code);
}
function modificar(){
 var origen=document.getElementById('origen').value;
 var destino=document.getElementById('destino').value;
 dd.getFile(origen,null,function(archivo){
  dd.getDirectory(destino,null,function(dir){
   archivo.moveTo(dir,null,exito,errores);
  },errores);
 },errores);
}
function exito(){
 document.getElementById('origen').value='';
 document.getElementById('destino').value='';
 mostrar();
}
function mostrar(){
 cajadatos.innerHTML='';
 dd.getDirectory(ruta,null,leerdir,errores);
}
function leerdir(dir){
 var lector=dir.createReader();
 var leer=function(){
  lector.readEntries(function(archivos){
   if(archivos.length){
    listar(archivos);
    leer();
   }
  }, errores);
 }
 leer();
}

function listar(archivos){
 for(var i=0; i<archivos.length; i++) {
  if(archivos[i].isFile) {
   cajadatos.innerHTML+=archivos[i].name+'<br>';
  }else if(archivos[i].isDirectory){
   cajadatos.innerHTML+='<span onclick="cambiardir(\''+archivos[i].
 name+'\')" class="directorio">+'+archivos[i].name+'</span><br>';
 }
 }
}
function cambiardir(nuevaruta){
 ruta=ruta+nuevaruta+'/';
 mostrar();
}
addEventListener('load', iniciar);
```

Código 16-13

Mover archivos.

En este último ejemplo, usamos funciones de códigos previos para crear o abrir nuestro sistema de archivos y mostrar el código de entradas en pantalla. La única función nueva en el Código 16-13 es modificar(). Esta función toma los valores de los campos origen y destino del formulario, y los utiliza para abrir el archivo original. Luego, si la operación es exitosa, abre el directorio de destino. Si ambas operaciones son exitosas, el método moveTo() es aplicado sobre el objeto file y el archivo es movido al directorio representado por dir. Si esta última operación es exitosa, la función exito() es llamada para vaciar los campos en el formulario y actualizar las entradas en pantalla ejecutando la función mostrar().

Hágalo usted mismo

Para probar este ejemplo necesita un archivo HTML con la plantilla del Código 16-12, el archivo CSS usado desde el comienzo de este capítulo, y un archivo llamado file.js con el Código 16-13. Recuerde subir los archivos a su servidor. Cree archivos y directorios en su sistema de archivos usando los códigos previos para tener entradas con las que trabajar. Utilice el formulario del último documento HTML para insertar los valores del archivo que será movido, con la ruta completa desde la raíz, y el directorio al cual será movido el archivo. Si el directorio se encuentra en la raíz del sistema de archivos no necesita usar barras, solo su nombre. Supongamos que usted desea mover un archivo llamado balance.txt al directorio declaraciones. Escriba el nombre del archivo (balance.txt) en el campo Origen, escriba el nombre del directorio (declaraciones) en el campo Destino y pulse el botón Guardar.

16.3.8 Copiar archivos

Por supuesto, la única diferencia entre el método moveTo() y el método copyTo() es que el último preserva el archivo original. Para usar el método copyTo(), solo debemos cambiar el nombre del método del Código 16-13. La función modificar() quedará como en el siguiente ejemplo.

```
function modificar(){
 var origen=document.getElementById('origen').value;
 var destino=document.getElementById('destino').value;

 dd.getFile(origen,null,function(archivo){
  dd.getDirectory(destino,null,function(dir){
   archivo.copyTo(dir,null,exito,errores);
  },errores);
 },errores);
}
```

Código 16-14

Copiar archivos.

Hágalo usted mismo

Reemplace la función modificar() del Código 16-13 con esta última y abra la plantilla del Código 16-12 para probar el código. Para copiar un archivo, debe repetir los pasos usados previamente para moverlo. Inserte la ruta del archivo a copiar en el campo Origen y la ruta del directorio donde desea generar la copia en el campo Destino.

16.3.9 Eliminar

Eliminar archivos y directorio es incluso más sencillo que mover y copiar. Todo lo que tenemos que hacer es obtener un objeto Entry del archivo o directorio que deseamos eliminar y aplicar el método remove() a esta referencia:

```
function modificar(){
 var origen=document.getElementById('origen').value;
 var origen=ruta+origen;
 dd.getFile(origen,null,function(entrada){
  entrada.remove(exito,errores);
 },errores);
}
```

Código 16-15

Eliminar archivos y directorios.

El Código 16-15 solo utiliza el valor del campo origen del formulario. Este valor, junto con el valor de la variable ruta, representará la ruta completa de la entrada que queremos eliminar. Usando este valor y el método getFile(), creamos un objeto Entry para la entrada y luego aplicamos el método remove().

Hágalo usted mismo

Reemplace la función modificar() del Código 16-13 con la nueva del Código 16-15. Esta vez solo necesita ingresar el valor del campo origen para especificar el archivo a ser eliminado.

Para eliminar un directorio en lugar de un archivo, el objeto Entry debe ser creado para ese directorio usando getDirectory(), pero el método remove() trabaja exactamente igual sobre un tipo de entrada u otro. Sin embargo, debemos tener en cuenta que si el directorio no está vacío, el método remove() retornará un error. Para eliminar un directorio y su contenido, todo al mismo tiempo, debemos usar otro método mencionado anteriormente llamado removeRecursively():

```
function modificar(){

var destino=document.getElementById('destino').value;

dd.getDirectory(destino,null,function(entrada){

entrada.removeRecursively(exito,errores);

},errores);

}
```

Código 16-16

Eliminar directorios no vacíos.

En la función del Código 16-16 usamos el valor del campo destino para indicar el directorio a ser eliminado. El método removeRecursively() eliminará el directorio y su contenido en una sola ejecución y llamará a la función exito() si la operación es realizada con éxito.

Hágalo usted mismo

Las funciones modificar() presentadas en el Código 16-14, el Código 16-15 y el Código 16-16 fueron construidas para reemplazar la misma función del Código 16-13. Para ejecutar estos ejemplos, utilice el Código 16-13, reemplace la función modificar() por la que quiere probar y abra la plantilla del Código 16-12 en su navegador. De acuerdo al método elegido, deberá ingresar uno o dos valores en el formulario.

16.4 Contenido de archivos

Además de la parte principal de la API File y la extensión ya estudiada, existe otra especificación llamada API File: Writer. Esta especificación declara nuevas interfaces para escribir y agregar contenido a archivos. Trabaja junto con el resto de la API combinando métodos y compartiendo objetos para lograr su objetivo.

16.4.1 Escribir contenido

Para escribir contenido en un archivo, necesitamos crear un objeto FileWriter, que es retornado por el método createWriter() de la interfaz FileEntry. Esta interfaz, añadida a la interfaz Entry, ofrece dos métodos para trabajar con archivos:

createWriter(éxito, error): Este método retorna un objeto FileWriter asociado con la entrada seleccionada.

file(éxito, error): Éste es un método que vamos a usar más adelante para leer el contenido del archivo. Crea un objeto File asociado con la entrada seleccionada (como el retornado por el elemento <input> o por una operación de arrastrar y soltar).

El objeto FileWriter, retornado por el método createWriter(), tiene sus propios métodos, propiedades y eventos para facilitar el proceso de agregar contenido a un archivo.

write(datos): Éste es el método que escribe contenido dentro del archivo. El contenido a ser insertado es provisto por el atributo datos en forma de blob.

seek(desplazamiento): Este método establece la posición del archivo en la cual el contenido será añadido. El valor del atributo desplazamiento debe ser declarado en bytes.

truncate(tamaño): Este método cambia el tamaño del archivo de acuerdo al valor del atributo tamaño (en bytes).

position: Esta propiedad retorna la posición en la cual ocurrirá la siguiente escritura. La posición será 0 para un nuevo archivo o diferente de 0 si algún contenido ha sido escrito dentro del archivo o si el método seek() ha sido aplicado previamente.

length: Esta propiedad retorna el largo del archivo.

writestat: Este evento es disparado cuando el proceso de escritura comienza.

progress: Este evento es disparado periódicamente para informar el progreso.

write: Este evento es disparado cuando los datos han sido completamente escritos en el archivo.

abort: Este evento es disparado si el proceso es abortado.

error: Este evento es disparado si ocurre un error en el proceso.

writeend: Este evento es disparado cuando el proceso termina.

Necesitamos crear un objeto más para preparar el contenido que va a ser agregado al archivo. El método write() solo usa blobs como atributos, por eso, para construir un blob, la API ofrece el constructor Blob():

Blob(matriz, propiedades): Este constructor toma una matriz con la información para el blob y devuelve un blob. La matriz puede contener cadenas de texto, otros blobs o datos con formato ArrayBuffer. El atributo propiedades es opcional, y es un objeto con dos posibles propiedades: type y endings. La propiedad type define el tipo mime de los datos de la matriz, mientras la propiedad endings define la codificación de los datos con dos valores posibles: transparent (los datos son usados tal como son) o native (la codificación de los datos es adaptada a la configuración del sistema operativo).

El documento HTML del Código 16-17 incorpora un segundo campo para insertar texto que representará el contenido del archivo. Será la plantilla utilizada en los próximos ejemplos:

```
<!DOCTYPE html>
<html lang="es">
<head>
 <title>API File</title>
 <link rel="stylesheet" href="file.css">
 <script src="file.js"></script>
</head>
<body>
 <section id="cajaformulario">
  <form name="formulario">
   <label for="entrada">Archivo:</label>

   <input type="text" name="entrada" id="entrada" required><br>
   <label for="texto">Texto: </label><br>
   <textarea name="texto" id="texto" required></textarea><br>
   <input type="button" id="boton" value="Aceptar">
  </form>
 </section>
 <section id="cajadatos">
  No hay información disponible
 </section>
</body>
</html>
```

Código 16-17

Formulario para ingresar el nombre del archivo y su contenido.

Para la escritura del contenido abrimos el sistema de archivos, obtenemos o creamos el archivo con getFile() e insertamos contenido en su interior con los valores ingresados en el formulario.

```
var cajadatos, dd;
function iniciar(){
 cajadatos=document.getElementById('cajadatos');
 var boton=document.getElementById('boton');
 boton.addEventListener('click', escribirarchivo);
 webkitRequestFileSystem(TEMPORARY, 5*1024*1024,creardd, errores);
}
function creardd(sistema){
 dd=sistema.root;
}
function errores(e){
 alert('Error: '+ e.code);
}
function escribirarchivo(){
 var nombre=document.getElementById('entrada').value;
 dd.getFile(nombre, {create: true, exclusive: false},
 function(entrada){
 entrada.createWriter(escribircontenido, errores);
 }, errores);
}
function escribircontenido(fileWriter) {
 var texto=document.getElementById('texto').value;
 fileWriter.addEventListener('writeend',exito);
 var blob=new Blob([texto]);
fileWriter.write(blob);
}
function exito(){
 document.getElementById('entrada').value='';
 document.getElementById('texto').value='';
 cajadatos.innerHTML='¡Hecho!';
}addEventListener('load', iniciar);
```

Código 16-18

Escribir contenido.

Cuando el botón Aceptar es presionado, la información en los campos del formulario es procesada por las funciones escribirarchivo() y escribircontenido(). La función escribirarchivo() toma el valor del campo entrada y usa getFile() para abrir o crear el archivo si no existe. El objeto Entry retornado es usado por createWriter() para crear un objeto FileWriter. Si la operación es exitosa, este objeto es enviado a la función escribircontenido().

La función escribircontenido() recibe el objeto FileWriter y, usando el valor del campo texto, escribe contenido dentro del archivo. El texto debe ser convertido en un blob antes de ser usado. Con este propósito, es creado un objeto BlobBuilder con el constructor BlobBuilder(), con el texto como atributo. Tome nota del uso de los corchetes dentro de los paréntesis para convertir el texto en una matriz. Finalmente, la información se encuentra en el formato apropiado para ser escrita dentro del archivo usando el método write().

Todo el proceso es asíncrono, lo que significa que el estado de la operación será constantemente informado a través de eventos. En la función escribircontenido(), solo detectamos el evento writeend para llamar a la función exito() y escribir el mensaje ¡Hecho! en la pantalla cuando la operación es finalizada. Sin embargo, usted puede seguir el progreso o controlar los errores aprovechando el resto de los eventos disparados por el objeto FileWriter.

Hágalo usted mismo

Copie la plantilla del Código 16-17 dentro de un nuevo archivo HTML (esta plantilla usa los mismos estilos CSS del Código 16-2). Cree un archivo Javascript llamado file.js con el Código 16-18. Abra el documento HTML en su navegador e inserte el nombre y el texto del archivo que quiere crear. Si el mensaje ¡Hecho! aparece en pantalla, todo funciona correctamente.

16.4.2 Agregar contenido

Debido a que no especificamos la posición en la cual el contenido debía ser escrito, el código previo simplemente escribirá el blob al comienzo del archivo. Para seleccionar una posición específica o agregar contenido al final de un archivo ya existente, es necesario usar previamente el método seek().

```
function escribircontenido(fileWriter) {
 var texto=document.getElementById('texto').value;
 fileWriter.seek(fileWriter.length);
 fileWriter.addEventListener('writeend', exito);
 var blob=new Blob([texto]);
 fileWriter.write(blob);
}
```

Código 16-19

Agregar contenido al final del archivo.

La función del Código 16-19 mejora la anterior función escribircontenido() incorporando un método seek() para mover la posición de escritura al final del archivo. De este modo, el contenido escrito por el método write() no sobrescribirá el contenido anterior.

Para calcular la posición del final del archivo en bytes, usamos la propiedad length. El resto del código es exactamente el mismo que en el Código 16-18.

16.4.3 Leer contenido

Es momento de leer lo que acabamos de escribir. El proceso de lectura usa técnicas de la parte principal de API File, estudiada al comienzo de este capítulo. Vamos a usar el constructor FileReader() y el método de lectura readAsText() para obtener el contenido del archivo.

```
var cajadatos, dd;

function iniciar(){
 cajadatos=document.getElementById('cajadatos');
 var boton=document.getElementById('boton');
 boton.addEventListener('click', leerarchivo);
 webkitRequestFileSystem(TEMPORARY, 5*1024*1024, creardd, errores);
}
function creardd(sistema){
 dd=sistema.root;
}
function errores(e){
alert('Error: '+e.code);
}
function leerarchivo(){
 var nombre=document.getElementById('entrada').value;
 dd.getFile(nombre, {create: false}, function(entrada) {
  entrada.file(leercontenido, errores);
}, errores);
}
function leercontenido(archivo){
 cajadatos.innerHTML='Nombre: '+archivo.name+'<br>';
 cajadatos.innerHTML+='Tipo: '+archivo.type+'<br>';
 cajadatos.innerHTML+='Tamaño: '+archivo.size+' bytes<br>';

 var lector=new FileReader();
 lector.addEventListener('load', exito);
 lector.readAsText(archivo);
} function exito(e){
 var resultado=e.target.result;
 cajadatos.innerHTML+='Contenido: '+resultado;
 document.getElementById('entrada').value='';
}
addEventListener('load', iniciar);
```

Código 16-20

Leyendo el contenido de un archivo en el sistema de archivos.

Hágalo usted mismo

Reemplace la función escribircontenido() del Código 16-18 por la nueva presentada en el Código 16-19 y abra el archivo HTML en su navegador. Inserte en el formulario el mismo nombre del archivo creado usando el código previo y el texto que quiere agregar al mismo.

Los métodos provistos por la interfaz FileReader para leer el contenido de un archivo, como readAsText(), requieren un blob o un objeto File como atributo. El objeto File representa el archivo a ser leído y es generado por el elemento <input> o una operación arrastrar y soltar. Como dijimos anteriormente, la interfaz FileEntry ofrece la opción de crear esta clase de objetos utilizando un método llamado file().

Cuando el botón Aceptar es presionado, la función leerarchivo() toma el valor del campo entrada del formulario y abre el archivo con ese nombre usando getFile(). El objeto Entry retornado por este método es representado por la variable entrada y es usado para generar el objeto File con el método file().

Debido a que el objeto File obtenido de este modo es el mismo que el generado

por el elemento <input> o la operación de arrastrar y soltar, todas las mismas propiedades usadas anteriormente están disponibles y podemos mostrar información básica acerca del archivo incluso antes de que el proceso de lectura del contenido comience. En la función leercontenido(), los valores de estas propiedades son mostrados en pantalla y el contenido del archivo es leído por el método readAsText.

El proceso de lectura es una réplica exacta del utilizado en el Código 16-3: el objeto FileReader es creado por el constructor FileReader(); la función exito(), establecida como detector del evento.

La función exito(), en lugar de imprimir una cadena de texto, tal como hacía antes, ahora muestra el contenido del archivo en la pantalla. Para hacer esto, tomamos el valor de la propiedad result perteneciente al objeto FileReader y lo declaramos como contenido del elemento cajadatos.

Hágalo usted mismo

El Código 16-20 utiliza solo el valor del campo entrada. Abra el archivo HTML con la última plantilla en su navegador e inserte el nombre del archivo que quiere leer. Debe ser un archivo que usted ya creó usando códigos previos o el sistema retornará un mensaje de error (create: false). Si el nombre del archivo es correcto, la información sobre este archivo y su contenido serán mostrados en pantalla.

16.5 Acceder a los archivos

El sistema de archivos que hemos estado usando todo este tiempo es solo un espacio reservado en el disco duro para nuestra aplicación. No podemos acceder a los archivos y directorios desde fuera del explorador, y no podemos mover las entradas a otro lugar en el disco duro.

Para resolver este problema, HTML5 introduce dos elementos: un método y un atributo.

toURL(): Este método es similar al método createObjectURL() estudiado en el Capítulo 9, pero es específico para una entrada de archivo. Se crea una URL que apunta al archivo que se puede utilizar más adelante como cualquier otra URL.

download: Éste es un nuevo atributo booleano para los elementos <a>. Permite informar al navegador que el recurso debe ser descargado.

```
 var cajadatos, dd, ruta;
 function iniciar(){
 cajadatos=document.getElementById('cajadatos');
 var boton=document.getElementById('boton');
 boton.addEventListener('click', crear);
 webkitRequestFileSystem(TEMPORARY, 5*1024*1024, creardd, errores);
}
function creardd(sistema){
 dd=sistema.root;
 ruta='';
 mostrar();
}
function errores(e){
alert('Error: '+e.code);
}
function crear(){
 var nombre = document.getElementById('entrada').value;
 if(nombre != ''){
 nombre = ruta + nombre;
 dd.getFile(nombre, {create: true, exclusive: false}, mostrar,
 errores);
 }

}
function mostrar(){
 document.getElementById('entrada').value='';
 cajadatos.innerHTML = '';
 dd.getDirectory(ruta, null, leerdir, errores);
}

function leerdir(dir){
 var lector = dir.createReader();
 var leer = function(){
 lector.readEntries(function(archivos){
 if(archivos.length){
 listar(archivos);
 leer();
 }
 }, errores);
 }
 leer();
}
function listar(archivos){
 var url;
 for(var i = 0; i < archivos.length; i++) {
  if(archivos[i].isFile) {
   url = archivos [i].toURL();
 cajadatos.innerHTML += '<a href="' + url + '" download>' +
 archivos[i].name + '</a><br>';
  }else if(archivos[i].isDirectory){
   cajadatos.innerHTML += '<span onclick="cambiardir(\'' +
 archivos[i].name + '\')" class="directory">' + archivos[i].nombre +
 '</span><br>';
 }
 }
}

function cambiardir(nuevaruta){
 ruta = ruta + nuevaruta + '/';
 mostrar();
}
addEventListener('load', iniciar);
```

Código 16-21

Descargar archivos del sistema de archivos al disco duro del usuario.

En este ejemplo, tomamos el Código 16-10 y modificamos ligeramente la función lista() para generar un enlace para cada archivo. La función obtiene la dirección URL del archivo aplicando el método toURL() para cada elemento de la matriz archivos y crea un enlace con el atributo download para que los usuarios puedan descargar los archivos con un clic.

Hágalo usted mismo

El Código 16-21 trabaja con el documento HTML del Código 16-7. Suba los dos archivos y los estilos CSS al servidor, y abra el documento en su navegador. Los archivos se descargaran a su disco duro al hacer clic en los enlaces.

Importante

No siempre es necesario aplicar el atributo llamado download. Hay casos en los que solo se requiere que el navegador muestre el contenido del archivo en la ventana, y si éste es su propósito, el atributo de descarga puede ser ignorado.

16.6 Sistema de archivos real

Siempre es bueno estudiar un caso de la vida real que nos permita entender el potencial de los conceptos aprendidos. Para finalizar este capítulo, vamos a crear una aplicación que combina varias técnicas de la API File con las posibilidades de manipulación de imágenes ofrecidas por la API Canvas.

Este ejemplo toma múltiples archivos de imagen y dibuja estas imágenes en el lienzo en una posición seleccionada al azar. Cada cambio efectuado en el lienzo es grabado en un archivo para lecturas posteriores, por tanto, el último trabajo realizado sobre el lienzo será mostrado en pantalla cada vez que acceda a la aplicación.

El documento HTML que vamos a crear es similar a la primera plantilla utilizada en este capítulo. Sin embargo, esta vez incluimos un elemento <canvas> dentro del elemento cajadatos:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>File API</title>
  <link rel="stylesheet" href="file.css">
  <script src="file.js"></script>
</head>
<body>
  <section id="cajaformulario">
    <form name="form">
     <label for="misarchivos">Images: </label>
     <input type="file" name="misarchivos" id="misarchivos" multiple>
    </form>

  </section>
  <section id="cajadatos">
      <canvas id="lienzo" width="500" height="350"></canvas>
  </section>
</body>
</html>
```

Código 16-22

Nueva plantilla con el elemento <canvas>.

El código de este ejemplo incluye métodos y técnicas de programación con las que ya está familiarizado, pero la combinación de especificaciones puede resultar confusa al principio. Veamos primero el código Javascript y analicemos luego cada parte, paso a paso:

```
var lienzo, dd;
function iniciar(){
 var elem = document.getElementById('lienzo');
 lienzo = elem.getContext('2d');
 var misarchivos = document.getElementById('misarchivos');
 misarchivos.addEventListener('change', proceso);
 webkitRequestFileSystem(TEMPORARY, 5*1024*1024, creardd,
        mostrarerror);
}
function creardd(fs){
 dd = fs.root;
 cargarlienzo();
}

function mostrarerror(e){
 alert('Error: ' + e.code);
}
function proceso(e){
 var archivos = e.target.archivos;
 for(var f = 0; f < archivos.length; f++){
  var archivo = archivos[f];
  if(archivo.type.match(/image.*/i)){
   var reader = new FileReader();
   reader.addEventListener('load', show);
   reader.readAsDataURL(archivo);
  }
 }
}

function show(e){
 var result = e.target.result;
 var image = document.createElement('img');
 image.setAttribute('src', result);
 image.addEventListener("load", function(){
  var x = Math.floor(Math.random() * 451);
  var y = Math.floor(Math.random() * 301);
  lienzo.drawImage(image, x, y, 100, 100);
  savecanvas();
 });
}

function cargarlienzo(){
 dd.getFile('lienzo.dat', {create: false}, function(entry) {
  entry.file(function(archivo){
   var reader = new FileReader();
   reader.addEventListener('load', function(e){
    var image = document.createElement('img');
    image.setAttribute('src', e.target.result);;
    image.addEventListener("load", function(){
     lienzo.drawImage(image, 0, 0);
    });
   });
   reader.readAsBinaryString(archivo);
  }, mostrarerror);
 }, mostrarerror);
}
addEventListener('load', iniciar);
```

Código 16-23

Utilizar una combinación de API File y API Canvas.

En este ejemplo trabajamos con dos API: API File (con sus extensiones) y API Canvas. En la función iniciar(), ambas API son inicializadas. El contexto para el lienzo es generado primero usando getContext(), y el sistema de archivos es solicitado después por el método requestFileSystem(). Como siempre, una vez que el sistema de archivos está listo, es llamada la función creardd() y la variable dd es inicializada en esta función con una referencia al directorio raíz del sistema de archivos. Esta vez, además, ha sido agregada una llamada a una nueva función al final de creardd() con el propósito de cargar el archivo que contiene la imagen generada por la aplicación la última vez que fue ejecutada.

Veamos en primer lugar cómo se construye la imagen grabada en el archivo mencionado. Cuando el usuario selecciona un nuevo archivo de imagen desde el formulario, el evento change es disparado por el elemento <input> y es llamada la función procesar(). Esta función toma los archivos enviados por el formulario, extrae cada objeto File de su matriz, controla si se trata de una imagen o no y, en caso positivo, lee el contenido de cada entrada con el método readAsDataURL(), retornando un valor en formato data:url.

Como puede ver, cada archivo es leído por la función procesar(), uno a la vez. Cada vez que una de estas operaciones es exitosa, el evento load es disparado y la función mostrar() es ejecutada. Como resultado, la función procesa cada imagen que el usuario haya seleccionado. La función mostrar() toma los datos del objeto lector, crea un objeto imagen con el constructor Image(), y asigna los datos leídos previamente como valor del atributo src de la imagen.

Al trabajar con imágenes debemos considerar el tiempo que la imagen tarda en ser cargada en la memoria. Por esta razón, luego de declarar la nueva fuente del objeto image, agregamos un detector del evento load que nos permitirá procesar la imagen solo cuando ésta haya sido completamente cargada. Una vez que este evento es disparado, la función anónima declarada para responder al evento es ejecutada. Esta función calcula una posición al azar para la imagen y la dibuja dentro del lienzo usando el método drawImage(). La imagen es reducida por este método a un tamaño fijo de 100×100 píxeles, sin importar el tamaño original (en la función mostrar() del Código 16-22).

Conceptos básicos

El método floor() pertenece al objeto nativo Match. Éste redondea el número entre paréntesis y devuelve un número entero.

Luego de que las imágenes seleccionadas son dibujadas, se llama a la función grabarlienzo(). Esta función se encargará de grabar el estado del lienzo cada vez que sea modificado, permitiendo a la aplicación recuperar el último trabajo realizado la siguiente vez que sea ejecutada. El método de API Canvas llamado toDataURL() es usado para retornar el contenido del lienzo como data:url. Para procesar estos datos, son realizadas varias operaciones dentro de la función grabarlienzo(). Primero, los datos en formato data:url son almacenados dentro de la variable info. Luego, se crea el archivo lienzo.dat (si aún no existe) o se abre con getFile() (si ya existe). Si esta operación es exitosa, el método llama a una función anónima en la que se crea el objeto FileWriter mediante el método createWriter(). Si esta operación es exitosa, este método también llama a una función anónima en la que el valor de la variable info (los datos sobre el estado actual del lienzo) son convertidos en un objeto blob y escritos en el archivo lienzo. dat por medio del método write().

Importante

En este ejemplo hemos tenido que obtener la información como un objeto data:url y convertirla en un blob con el constructor Blob() antes de que pudiera ser utilizado como contenido del archivo. Este proceso podría haber sido simplificado con la aplicación del método toBlob() en lugar del método toDataURL(). El método toBlob() devuelve un objeto blob en lugar de un objeto data:url, pero en el momento de publicar este manual, el método aún no ha sido implementado por los navegadores.

Bien, es momento de volver a la función cargarlienzo(). Como ya mencionamos, esta función es llamada por la función creardd() tan pronto como es cargada la aplicación. Tiene el propósito de leer el archivo con el resultado anterior y dibujarlo en pantalla. A este punto usted ya sabe de qué archivo estamos hablando y cómo es generado, veamos entonces cómo esta función restaura el último trabajo realizado sobre el lienzo.

Una vez es llamada la función cargarlienzo(), carga el archivo lienzo.dat. Si el archivo no existe, el método getFile() retornará un error, pero si lo encuentra, el método ejecuta una función anónima que tomará la entrada y usará el método file() para generar un objeto File con estos datos. Este método, si es exitoso, también ejecuta una función anónima para leer el archivo y obtener su contenido como datos binarios usando readAsBinaryString(). El contenido obtenido, como ya sabemos, es una cadena de texto en formato data:url que debe ser asignada como fuente de una imagen antes de ser dibujada en el lienzo. Por este motivo, lo que hacemos dentro de la función anónima llamada por el evento load una vez que el archivo es completamente cargado, es crear un objeto imagen, declarar los datos obtenidos como la fuente de esta imagen, y (cuando la imagen es completamente cargada) dibujarla en el lienzo con drawImage().

El resultado obtenido por esta pequeña pero interesante aplicación es sencillo:

las imágenes seleccionadas desde el elemento <input> son dibujadas en el lienzo en una posición al azar y el estado del lienzo es conservado en un archivo. Si se cierra el navegador, no importa por cuánto tiempo, el archivo será leído la próxima vez que la aplicación sea usada, el estado previo del lienzo es restaurado y el último trabajo vuelve a mostrarse, como si nunca hubiese sido abandonado. No es un ejemplo realmente útil, pero se puede apreciar su potencial.

Hágalo usted mismo

Usando la API Drag and Drop puede arrastrar y soltar archivos de imagen dentro del lienzo en lugar de cargar las imágenes desde el elemento <input>. Intente combinar el Código 16-22 con algunos códigos del Capítulo 13 para integrar estas API.

17 API Geolocation

17.1 Encontrar su lugar

La API Geolocation fue diseñada para que los navegadores puedan proveer un mecanismo de detección por defecto que permita a los desarrolladores determinar la ubicación física real del usuario. Previamente solo contábamos con la opción de construir una gran base de datos con información sobre direcciones IP y programar códigos exigentes dentro del servidor que nos darían una idea aproximada de la ubicación del usuario (generalmente tan imprecisa como su país).

Esta API aprovecha nuevos sistemas, como triangulación de red y GPS, para retornar una ubicación precisa del dispositivo que está accediendo a la aplicación. La valiosa información retornada nos permite construir aplicaciones que se adaptarán a las necesidades particulares del usuario o proveerán información localizada de forma automática. Tres métodos específicos son provistos para usar la API:

getCurrentPosition(ubicación, error, configuración): Éste es el método utilizado para consultas individuales. Puede recibir hasta tres atributos: una función para procesar la ubicación retornada, una función para procesar los errores retornados, y un objeto para configurar cómo la información será adquirida. Solo el primer atributo es obligatorio para que el método trabaje correctamente.

watchPosition(ubicación, error, configuración): Este método es similar al anterior, excepto que comenzará un proceso de vigilancia para la detección de nuevas ubicaciones. Trabaja de forma similar que el conocido método setInterval() de Javascript, repitiendo el proceso automáticamente en determinados períodos de tiempo de acuerdo a la configuración por defecto

o a los valores de sus atributos.

clearWatch(id): El método watchPosition() retorna un valor que puede ser almacenado en una variable para luego ser usado como referencia por el método clearWatch() y así detener la vigilancia. Este método es similar a clearInterval(), usado para detener los procesos comenzados por setInterval().

17.1.1 Documento HTML

El que sigue va a ser nuestro documento para este capítulo. Es extremadamente sencillo, solo con un botón dentro del elemento <section> que se va a utilizar para mostrar la información recuperada por el sistema de localización.

```
<!DOCTYPE html>
<html lang="es">
<head>
 <title>Geolocation</title>
 <script src="geolocation.js"></script>

</head>
<body>
 <section id="ubicacion">
 <input type="button" id="obtener" value="Obtener mi Ubicación">
 </section>
</body>
</html>
```

Código 17-1

Documento HTML para la API Geolocation.

17.1.2 getCurrentPosition(ubicación)

Como dijimos, solo el primer atributo es requerido para que trabaje correctamente el método getCurrentPosition(). Este atributo es una función que recibirá un objeto llamado Position, el cual contiene toda la información retornada por los sistemas de ubicación. El objeto Position tiene dos propiedades:

coords: Esta propiedad contiene un objeto con una serie de propiedades que proporcionan la ubicación del dispositivo y otros datos importantes. Las propiedades disponibles son: latitude (latitud), longitude (longitud), altitude (altitud en metros), accuracy (exactitud en metros), altitudeAccuracy (exactitud de la altitud en metros), heading (dirección en grados) y speed (velocidad en metros por segundo).

timestamp: Este atributo indica el momento en el que la información fue adquirida (en formato timestamp).

```
function iniciar(){
 var boton=document.getElementById('obtener');
 boton.addEventListener('click', obtener);
}
function obtener(){
 navigator.geolocation.getCurrentPosition(mostrar);
}
function mostrar(posicion){
 var ubicacion=document.getElementById('ubicacion');
 var datos='';
 datos+='Latitud: '+posicion.coords.latitude+'<br>';
 datos+='Longitud: '+posicion.coords.longitude+'<br>';
 datos+='Exactitud: '+posicion.coords.accuracy+'mts.<br>';
 ubicacion.innerHTML=datos;
}
addEventListener('load', iniciar);
```

Código 17-2

Obtener información sobre la localización del usuario.

Implementar la API Geolocation es una tarea muy sencilla: declaramos el método getCurrentPosition() y creamos una función que mostrará los valores retornados por el mismo. El método getCurrentPosition() es un método del objeto geolocation. Este nuevo objeto, que forma parte del objeto Navigator, es un objeto Javascript que, por su parte, fue implementado para retornar información acerca del navegador y el sistema. Por lo tanto, para acceder al método getCurrentPosition() la sintaxis a usar es la siguiente: navigator.geolocation.getCurrentPosition(función), donde función es una función personalizada que recibirá el objeto Position y procesará la información que contiene.

En el Código 17-2 llamamos a esta función personalizada mostrar(). Cuando el método getCurrentPosition() es llamado, es creado un nuevo objeto Position con la información de la ubicación actual del usuario y es enviado a la función mostrar(). Hacemos referencia a este objeto dentro de la función con la variable posicion, y luego usamos esta variable para mostrar los datos.

El objeto Position tiene dos importantes atributos: coords y timestamp. En nuestro ejemplo solo usamos coords para acceder a la información que queremos mostrar (latitud, longitud y exactitud). Estos valores son grabados en la variable datos y luego mostrados en la pantalla como el nuevo contenido del elemento ubicacion.

Hágalo usted mismo

Cree dos archivos con el Código 9-1 y el Código 9-2, suba los archivos a su servidor y luego abra el documento HTML en su navegador. Cuando haga clic en el botón, el navegador le preguntará si desea activar o no el sistema de ubicación geográfica para esta aplicación. Si le permite a la aplicación acceder a esta información, entonces su ubicación, incluyendo longitud, latitud y exactitud, será mostrada en pantalla.

17.2.3 getCurrentPosition(ubicación, error)

¿Y si permite al navegador acceder a la información acerca de su ubicación? Agregando un segundo atributo al método getCurrentPosition(), otra función, podremos capturar los errores producidos en el proceso. Uno de esos errores, por supuesto, ocurre cuando el usuario no acepta compartir sus datos.

Junto con el objeto Position, el método getCurrentPosition() retorna el objeto PositionError si es detectado un error. Este objeto tiene dos atributos internos: error y message, que proporcionan el valor y la descripción del error. Los tres posibles errores son representados por las siguientes constantes:

PERMISSION_DENIED (permiso denegado): valor 1. Este error ocurre cuando el usuario no acepta activar el sistema de ubicación para compartir su información.

POSITION_UNAVAILABLE (ubicación no disponible): valor 2. Este error ocurre cuando la ubicación del dispositivo no pudo determinarse.

TIMEOUT (tiempo excedido): valor 3. Este error ocurre cuando la ubicación no pudo ser determinada en el período de tiempo máximo declarado en la configuración.

```
function iniciar(){
 var boton=document.getElementById('obtener');
 boton.addEventListener('click', obtener);
}
function obtener(){
 navigator.geolocation.getCurrentPosition(mostrar, errores);
}
function mostrar(posicion){
 var ubicacion=document.getElementById('ubicacion');
 var datos='';
 datos+='Latitud: '+posicion.coords.latitude+'<br>';
 datos+='Longitud: '+posicion.coords.longitude+'<br>';
 datos+='Exactitud: '+posicion.coords.accuracy+'mts.<br>';
 ubicacion.innerHTML=datos;
}
function errores(error){
 alert('Error: '+error.code+' '+error.message);
}
addEventListener('load', iniciar);
```

Código 17-3

Mostrar mensajes de error.

Los mensajes de error son ofrecidos para uso interno. El propósito es ofrecer un mecanismo para que la aplicación reconozca la situación y proceda de acuerdo al error recibido. En el Código 17-3, agregamos un segundo atributo al método getCurrentPosition() (otra función) y creamos la función errores() para mostrar la información de los atributos code y message. El valor de code será un número entero entre 0 y 3 de acuerdo al número de error (listado anteriormente).

El objeto PositionError es enviado a la función errores() y representado en esta función por la variable error. También podríamos haber controlado los errores de forma individual (error.PERMISSION_DENIED, por ejemplo) y mostrar una alerta solo para el error en particular que hemos seleccionado.

17.2.4 getCurrentPosition(ubicación, error, configuración)

El tercer atributo que podemos usar en el método getCurrentPosition() es un objeto que contiene hasta tres posibles propiedades:

enableHighAccuracy: Ésta es una propiedad booleana para informar al sistema que requerimos de la información más exacta que nos pueda ofrecer. El navegador intentará obtener esta información a través de sistemas como GPS, por ejemplo, para retornar la ubicación exacta del dispositivo. Estos son sistemas que consumen muchos recursos, por lo que su uso debería estar limitado a circunstancias muy específicas. Para evitar consumos innecesarios, el valor por defecto de esta propiedad es false (falso).

timeout: Esta propiedad indica el tiempo máximo de espera para que la operación finalice. Si la información de la ubicación no es obtenida antes del tiempo indicado, es retornado el error TIMEOUT. Su valor se expresa en milisegundos.

maximumAge: Las ubicaciones encontradas previamente son almacenadas en un caché en el sistema. Si consideramos apropiado recurrir a la información grabada en lugar de intentar obtenerla desde el sistema (para evitar consumo de recursos o para una respuesta rápida), esta propiedad puede ser declarada con un tiempo límite específico. Si la última ubicación almacenada es más antigua que el valor de este atributo, entonces se solicita al sistema una nueva ubicación. Su valor es en milisegundos.

```
function iniciar(){
 var boton=document.getElementById('obtener');
 boton.addEventListener('click', obtener);
}
function obtener(){
 var geoconfig={
 enableHighAccuracy: true,
 timeout: 10000,
 maximumAge: 60000
 };
 navigator.geolocation.getCurrentPosition(mostrar, errores,geoconfig);
}
function mostrar(posicion){
 var ubicacion=document.getElementById('ubicacion');
 var datos='';
 datos+='Latitud: '+posicion.coords.latitude+'<br>';
 datos+='Longitud: '+posicion.coords.longitude+'<br>';
 datos+='Exactitud: '+posicion.coords.accuracy+'mts.<br>';
 ubicacion.innerHTML=datos;
}
function errores(error){
 alert('Error: '+error.code+' '+error.message);
}
addEventListener('load', iniciar);
```

Código 17-4

Configuración del sistema.

El Código 17-4 intentará obtener la ubicación más exacta posible del dispositivo en no más de 10 segundos, pero solo si no hay una ubicación previa en el caché capturada menos de 60 segundos atrás (si existe una ubicación previa con menos de 60 segundos de antigüedad, éste será el objeto Position retornado).

El objeto que contiene los valores de configuración es almacenado en la variable geoconfig y esta variable luego es usada como el tercer atributo del método getCurrentPosition(). No hemos cambiado nada más en el resto del código. La función mostrar() mostrará la información en la pantalla independientemente de su origen (es decir, si proviene del caché o es nueva).

Con el último código, podemos apreciar el propósito real de la API Geolocation y cuál fue la intención de sus desarrolladores. Las funciones más efectivas y prácticas están orientadas hacia dispositivos móviles. El valor true (verdadero) para la propiedad enableHighAccuracy, por ejemplo, le solicitará al navegador usar sistemas como GPS para obtener la ubicación más exacta posible (un sistema casi exclusivo de los dispositivos móviles), y los métodos watchPosition() y clearWatch(), que veremos a continuación, trabajan sobre ubicaciones actualizadas constantemente, algo solo posible, por supuesto, cuando el dispositivo que está accediendo la aplicación es móvil (y se está moviendo).

Esto pone en relevancia a dos asuntos importantes. Primero, la mayoría de nuestros códigos tendrán que ser probados en un dispositivo móvil para saber

exactamente cómo trabajan en una situación real. Y segundo, deberemos ser responsables con el uso de esta API. GPS y otros sistemas de localización consumen muchos recursos y en la mayoría de los casos pueden acabar pronto con la batería del dispositivo si no somos cautelosos.

Con respecto al primer punto, disponemos de una alternativa. Simplemente visite el enlace http://dev.w3.org/geo/api/test-suite/ y lea acerca de cómo experimentar y probar Geolocation API. Con respecto al segundo punto, solo un consejo: configure la propiedad enableHighAccuracy como true únicamente cuando sea estrictamente necesario y no abuse de esta posibilidad.

17.2.5 watchPosition(ubicación, error, configuración)

Similar a getCurrentPosition(), el método watchPosition() recibe tres atributos y realiza la misma tarea: obtiene la ubicación del dispositivo que está accediendo a la aplicación. La única diferencia es que el primero realiza una única operación, mientras que watchPosition() ofrece nuevos datos cada vez que la ubicación cambia. Este método vigilará todo el tiempo la ubicación y enviará información a la función correspondiente cuando se detecte una nueva ubicación, a menos que cancelemos el proceso con el método clearWatch().

Éste es un ejemplo de cómo implementar el método watchPosition() basado en códigos previos:

```
function iniciar(){
 var boton=document.getElementById('obtener');
 boton.addEventListener('click', obtener);
}
function obtener(){
 var geoconfig={
 enableHighAccuracy: true,
 maximumAge: 60000
 };
 control=navigator.geolocation.watchPosition(mostrar, errores,
                geoconfig);
}
function mostrar(posicion){
 var ubicacion=document.getElementById('ubicacion');
 var datos='';
 datos+='Latitud: '+posicion.coords.latitude+'<br>';
 datos+='Longitud: '+posicion.coords.longitude+'<br>';
 datos+='Exactitud: '+posicion.coords.accuracy+'mts.<br>';
 ubicacion.innerHTML=datos;
}
function errores(error){
 alert('Error: '+error.code+' '+error.message);
}
addEventListener('load', iniciar);
```

Código 17-5

Prueba del método watchPosition().

Si prueba este código en un ordenador de escritorio, no notará ningún cambio, pero en un dispositivo móvil será mostrada nueva información cada vez que haya una modificación en la ubicación del dispositivo. El atributo maximumAge determina qué tan seguido será enviada la información a la función mostrar(). Si la nueva ubicación es obtenida 60 segundos (60000 milisegundos) luego de la anterior, entonces será mostrada, en caso contrario la función mostrar() no será llamada.

Note que el valor retornado por el método watchPosition() es almacenado en la variable control. Esta variable es como un identificador de la operación. Si más adelante queremos cancelar el proceso de vigilancia, solo debemos ejecutar la línea clearWatch(control) y watchPosition() dejará de actualizar la información.

17.2.6 Usos prácticos con Google Maps

Hasta el momento hemos mostrado la información sobre la ubicación exactamente como la recibimos. Sin embargo, estos valores normalmente no significan nada para la gente común. La mayoría de nosotros no podemos decir inmediatamente cuál es nuestra ubicación actual en valores de latitud y longitud, y mucho menos identificar a partir de estos valores una ubicación en el mundo.

Disponemos de dos alternativas: usar esta información internamente para calcular posiciones, distancias y otros valores que nos permitirán ofrecer resultados específicos a nuestros usuarios (como productos o restaurantes en el área), o podemos ofrecer la información obtenida por medio de la API Geolocation en un medio mucho más comprensible. ¿Y qué más comprensible que un mapa para representar una ubicación geográfica?

Ya hemos mencionado en este mismo manual la API Google Maps. Ésta es una API Javascript externa proporcionada por Google, que nada tiene que ver con HTML5 pero es ampliamente utilizada en sitios webs y aplicaciones de nuestros días. Ofrece una variedad de alternativas para trabajar con mapas interactivos e incluso vistas reales de lugares muy específicos a través de la tecnología StreetView.

Vamos a mostrar un ejemplo simple en el que sacaremos provecho de una parte de la API llamada Static Maps API.

Con esta API específica, solo necesitamos construir una URL con la información de la ubicación para obtener en respuesta la imagen de un mapa con el área

seleccionada.

```
function iniciar(){
 var boton=document.getElementById('obtener');
 boton.addEventListener('click', obtener);
}
function obtener(){
 navigator.geolocation.getCurrentPosition(mostrar, errores);
}
function mostrar(posicion){
 var ubicacion=document.getElementById('ubicacion');
 var mapurl='http://maps.google.com/maps/api/staticmap?center=' +
       posicion.coords.latitude + ',' + posicion.coords.longitude
       + '&zoom=12&size=400x400&sensor=false&markers=' + posicion.
       coords.latitude + ',' + posicion.coords.longitude;
  ubicacion.innerHTML='<img src="'+mapurl+'">';
}

function errores(error){
 alert('Error: '+error.code+' '+error.message);
}
addEventListener('load', iniciar);
```

Código 17-6

Representar la ubicación en un mapa.

El código es simple. Usamos el método getCurrentPosition() y enviamos la información a la función mostrar() como siempre. Sin embargo, en esta función los valores del objeto Position son agregados a una URL de Google y luego la dirección obtenida es insertada como fuente de un elemento <img> para mostrar el mapa en pantalla.

Hágalo usted mismo

Pruebe el Código 17-6 en su navegador usando la plantilla creada con el Código 17-1. Cambie los valores de los atributos zoom y size en la URL para modificar el mapa retornado por la API. Visite la página de Google Maps API para estudiar las diferentes alternativas provistas por esta API: code.google.com/apis/maps/. Cambie los valores de los atributos enfocar y tamaño en la URL para modificar el mapa devuelto por la API.

18 API History

18.1 La interfaz de API History

La API History en HTML5 es, de hecho, una versión mejorada de la antigua API, que no había sido implementada oficialmente, pero que funcionó en diversos navegadores por mucho tiempo. Esta API disponía de un grupo de métodos y propiedades, algunos de ellos como parte del objeto History. La nueva API History mejoró dicho objeto y se incluyó en la especificación HTML5 oficial como la interfaz de API History. Esta interfaz combina todos los métodos y propiedades anteriores con algunos nuevos, para usar y modificar el historial del navegador de acuerdo a nuestras necesidades.

18.1.1 Navegar por la Web

Dentro del navegador el historial muestra una lista de todas las páginas web (URL) visitadas por el usuario durante una sesión. Es lo que hace que la navegación sea posible. Con los botones de navegación que se encuentran en la barra de navegación de todo navegador podemos ir hacia atrás o hacia adelante en esta lista y ver todos los documentos anteriores. Esta lista se basa en las direcciones URL reales generadas por los sitios web incluidas en cada hipervínculo dentro de los documentos. Con las flechas del navegador podemos cargar una página que hemos visitado antes o volver a la última visitada.

A pesar de la funcionalidad de los botones de navegación, a veces conviene navegar por el historial desde dentro del documento. Para imitar las flechas de navegación, en Javascript disponemos de los siguientes métodos y propiedades:

back(): Este método lleva al navegador un paso atrás en el historial de la sesión (imitando la flecha izquierda del navegador).

forward(): Este método lleva al navegador un paso hacia delante en el historial de la sesión (imitando la flecha derecha del navegador).

go(pasos): Este método lleva al navegador a avanzar o retroceder según los pasos indicados en el historial de la sesión. El atributo pasos puede tener un valor negativo o positivo de acuerdo con la dirección que se tome.

length: Esta propiedad recupera el número de entradas en el historial de la

sesión (el total de direcciones URL de la lista).

Todo lo anterior pertenece al objeto History, que es parte del objeto Window. Tenemos que declararlos como parte del objeto History con una expresión similar a: history.back(). También podemos mencionar el objeto Window pero, como hemos explicado antes, esto no es necesario. Por ejemplo, si queremos volver a la página web anterior, se puede utilizar la línea window.history.back() o window.history.go(-1).

Importante

Esta parte de la API ya es conocida y utilizada por la mayoría de los diseñadores y programadores web hoy en día. No vamos a ver ningún ejemplo de estos métodos, pero siempre que lo desee puede visitar nuestro sitio web y seguir los enlaces de este capítulo para obtener más información.

18.1.2 Nuevos métodos

Con el tiempo, el uso del objeto XMLHttpRequest se ha convertido en estándar y las aplicaciones Ajax tuvieron un éxito extraordinario. Cambió la forma en la que la gente navega y accede a los documentos en Internet. Ahora es una práctica común el programar pequeños códigos para recuperar la información de los servidores y mostrarla dentro del documento en el que se está trabajando sin actualizar la página completa. Los usuarios interactúan con sitios web modernos y aplicaciones desde la misma URL, reciben información, introducen datos y obtienen resultados de procesos en la misma página web. La Red ha comenzado a emular las aplicaciones de escritorio. Sin embargo, los navegadores realizan el seguimiento de la actividad del usuario a través de las direcciones URL. Dichas direcciones constituyen, de hecho, los datos del historial de navegación, ya que indican dónde se encuentra el usuario en un determinado momento. Como las nuevas herramientas web evitan usar las URL para localizar la ubicación del usuario en la Red, hay pasos importantes que se pierden en el proceso sin dejar rastro. Los usuarios pueden actualizar sus datos docenas de veces en Internet y sin embargo no dejar rastro en la lista del historial para indicar los pasos seguidos y ayudarles a ir hacia atrás o hacia adelante.

Se incorporaron nuevos métodos y propiedades a la API History con la intención de modificar manualmente la dirección URL en la barra de direcciones y la lista del historial utilizando simplemente el código Javascript. A partir de ahora tenemos la posibilidad de añadir URL falsas al historial y mantener la mirada puesta en la actividad del usuario.

PushState (estado, título, url): Este método crea una nueva entrada en el historial de la sesión. El atributo estado declara un valor para el estado de la entrada. Es útil para identificar la entrada más adelante y puede ser especificado como una cadena u objeto JSON. El atributo título es el título de la entrada y el atributo url es la URL de la entrada que estamos

generando (este valor sustituirá a la URL que aparece en la barra de direcciones).

replaceState (estado, título, url): Este método funciona exactamente igual que pushState (), pero no se genera una nueva entrada. En su lugar se sustituye la información por la más actual.

state: Esta propiedad devuelve el valor del estado de la entrada actual, el cual será nulo a menos que sea declarado por uno de los métodos anteriores a través del atributo estado.

18.1.3 URL falsas

Las URL generadas usando métodos tales como pushState() constituyen direcciones URL falsas en el sentido de que los navegadores nunca comprueban la validez de estas direcciones y la existencia del documento que se está señalando. Nos toca a nosotros el asegurarnos de que estas URL falsas sean de hecho válidas y útiles.

Para crear una nueva entrada en el historial del navegador y cambiar la dirección URL en la barra de direcciones, tenemos que utilizar el método pushState(). Veamos un ejemplo de cómo funciona esto:

```
<!DOCTYPE html>
<html lang="en">
<head>
 <title>API History</title>
 <link rel="stylesheet" href="history.css">
 <script src="history.js"></script>
</head>
<body>
 <section id="principal">
  Éste es el contenido inicial<br>
  <span id="url">Página 2</span>
 </section>
 <aside id="datos"></aside>
</body>
</html>
```

Código 18-1

Plantilla HTML básica para probar la API History.

En el Código 18-1 tenemos una plantilla HTML con los elementos básicos necesarios para probar la API History. Hay un contenido permanente dentro del elemento <section>, que es identificado como principal, un texto que se convertirá en un enlace para generar una página virtual secundaria de la página web y un elemento datos, que contendrá el contenido alternativo.

Vamos a incluir los estilos básicos necesarios para distinguir las partes de la plantilla del documento.

```
#principal{
 float: left;
 padding: 20px;
 border: 1px solid #999999;
}
#datos{
 float: left;
 width: 500px;
 margin-left: 20px;
 padding: 20px;
 border: 1px solid #999999;
}
#principal span{
 color: #0000FF;
 cursor: pointer;
}
```

Código 18-2

Estilos para las cajas y los elementos .

Conceptos básicos

La propiedad CSS cursor se utiliza para especificar el gráfico que se mostrará para representar el puntero del ratón. El sistema cambia automáticamente el cursor de acuerdo al contenido sobre el cual se encuentre el puntero del ratón. Para el contenido general, por ejemplo, el puntero se representa como una flecha, para el texto como una barra vertical y para los enlaces como una pequeña mano que señala en el propio enlace. En el Código 18-2 se cambia el cursor al final para mostrar al usuario que el contenido es interactivo. Hay varios valores disponibles para esta propiedad. Para obtener más información, visite nuestra web y siga los enlaces de este capítulo.

Lo que vamos a hacer en este ejemplo es añadir una nueva entrada con el método pushState() y actualizar el contenido sin actualizar la página o cargar otro documento.

```
function iniciar(){
 databox = document.getElementById('datos');
 url = document.getElementById('url');
 url.addEventListener('click', cambiarpag);
}
function cambiarpag(){
 databox.innerHTML = 'La URL es la página 2';
 history.pushState(null, null, 'pag2.html');
}

addEventListener('load', iniciar);
```

Código 18-3

Generar una nueva dirección URL y su contenido (history.js).

En la función iniciar() del Código 18-3 se crea la referencia adecuada al elemento datos y se añade un detector para el evento click del elemento . Cada vez que el usuario haga clic en el texto dentro de , se llamará a la función cambiarpag(). La función cambiarpag() lleva a cabo dos tareas: actualiza el contenido de la página con información nueva e inserta una nueva URL en el historial. Después de que se ejecuta la función, datos muestra el texto La URL es la página 2, y la URL del documento principal es sustituida en la barra de direcciones por la URL falsa pag2.html.

Hágalo usted mismo

Copie el Código 18-1 en un archivo HTML, cree un archivo CSS llamado history.css con el Código 18-2 y un archivo Javascript denominado history.js con el Código 18-3. Súbalos al servidor y abra el archivo HTML en su navegador. Haga clic en el texto Página 2 y compruebe cómo la URL cambia a la falsa, generada por el código.

Los atributos estado y título para el método pushState()se declaran como null en este proceso. El atributo título no está siendo utilizado por ningún navegador actualmente y siempre debe ser declarado como null; por el contrario el atributo estado lo vamos a aplicar y utilizar en los siguientes ejemplos dada su utilidad.

18.1.4 La propiedad state

Lo que hemos hecho hasta ahora es manipular el historial de la sesión. Hicimos “creer” al navegador que el usuario visitaba una URL que no existía.

Después de que el usuario hiciera clic en la Página 2 del enlace, la URL falsa llamada pag2.html se mostraba en la barra de direcciones y se introducían contenidos nuevos en datos, todo sin necesidad de refrescar la página web o de cargar otra página. Se trataba de un buen truco, pero que no estaba completo. De momento el navegador no toma en cuenta la dirección URL como un documento real. Si intentamos avanzar o retroceder en el historial utilizando los botones de navegación del navegador, la URL cambiará entre la que se genera artificialmente y la del documento principal, pero el contenido del documento no cambia en absoluto. Tenemos que detectar cuándo las URL falsas son visitadas de nuevo y además realizar las modificaciones adecuadas en el documento para mostrar el estado que corresponda a la URL actual.

Hemos mencionado antes de la existencia de la propiedad state. El valor de esta propiedad puede ser declarado durante la generación de una nueva URL, y de este modo se puede usar para identificar cuál es la dirección web actual. Para trabajar con esta propiedad, la API proporciona un nuevo evento:

popstate: Este evento se dispara en determinadas circunstancias, cuando se vuelve a visitar una URL o una vez que el documento se ha cargado. Provee la propiedad state con el valor del estado declarado cuando la URL ha sido genera por medio de los métodos pushState() o replaceState(). Este valor es null cuando la URL es real, a menos que lo cambiemos usando replaceState(), como veremos a continuación.

En el siguiente código vamos a mejorar lo que ya se mostró en el ejemplo anterior mediante la aplicación del popstate y el replaceState() para detectar cuál URL solicita el usuario en cada momento.

```
function iniciar(){
 databox = document.getElementById('datos');
 url = document.getElementById('url');
 url.addEventListener('click', cambiarpag);
 addEventListener('popstate', nuevaurl);
 history.replaceState(1, null);
}
function cambiarpag(){
 mostrar(2);
 history.pushState(2, null, 'pag2.html');
}
function nuevaurl(e){
 mostrar(e.state);
}
function mostrar(current){
 databox.innerHTML = 'La URL es la página ' + current;
}

addEventListener('load', iniciar);
```

Código 18-4

Hacer un seguimiento de la posición del usuario (history.js).

Hay dos cosas que tenemos que hacer para tener un control total del proceso. En primer lugar, debemos declarar un valor de estado para cada URL que se va a utilizar, tanto para las URL falsas como para las reales. Y en segundo lugar, hay que actualizar el contenido del documento de acuerdo con la URL actual.

En la función iniciar() que se muestra en el Código 18-4, se agrega un detector para el evento popstate. Así, cuando se visita una URL nuevamente, se ejecuta la función nuevaurl(). Esta función simplemente actualiza el contenido de datos con un mensaje que informa cuál es la URL donde se está en ese momento, toma el valor de la propiedad state y lo envía a la función mostrar() para que se muestre en pantalla.

Lo descrito anteriormente funcionará para las URL falsas, pero tal como hemos dicho antes, las URL reales no tienen un valor state por defecto. Mediante el uso del método replaceState() al final de la función iniciar() podemos cambiar la información de la entrada actual (la URL real para el documento principal) y asignar al estado un valor de 1. Así, cada vez que un usuario visita de nuevo el documento principal, se puede detectar comprobando este valor.

La función cambiarpag() es la misma, salvo que esta vez utiliza la función mostrar() para actualizar el contenido del documento y declarar el valor 2 para el estado de la URL falsa.

La aplicación funciona de la siguiente manera: cuando el usuario hace clic en el enlace Página 2, aparece el siguiente mensaje en pantalla: La URL es la página 2 y la URL es reemplazada en la barra de direcciones por pag2.html (incluyendo la ruta completa, por supuesto). Esto es lo que hemos hecho hasta el momento, pero a partir de aquí el asunto se vuelve más interesante. Si el usuario pulsa la flecha izquierda en la barra de navegación, la dirección URL cambiará por la que se encuentra en una posición anterior en el historial (que es la URL real de nuestro documento), y el evento popstate se disparará. Dicho evento llama a la función nuevaurl(), la cual lee el valor de la propiedad state y la envía a la función mostrar(). Ahora el valor del estado pasa a ser 1 (que es el valor que usamos para esta URL a través del método replaceState()), y el mensaje que se muestra en la pantalla será La URL es la página 1. Si el usuario vuelve a la URL falsa usando la flecha derecha en la barra de navegación, el valor del estado será 2, y el mensaje que se muestra en pantalla volverá a ser La URL es la página 2.

Como se puede ver, el valor de la propiedad state es cualquier valor que desee usar para controlar cuál es la URL actual; así podremos adaptar el contenido del documento a la misma.

Hágalo usted mismo

Hágalo usted mismo: Utilice el Código 18-1 y el Código 18-2 para crear el documento HTML y los estilos CSS. Copie el contenido del Código 18-4 en el archivo history.js y suba todos los archivos a su servidor. Abra el documento HTML en su navegador y haga clic en el texto Página 2. La dirección URL y el contenido de datos cambiarán de acuerdo con la URL correspondiente. Haga clic en las flechas derecha e izquierda del navegador varias veces para moverse en el historial y verificar si la dirección URL cambia y si el contenido se actualiza en pantalla de acuerdo con la URL que se seleccione.

Importante

La URL pag2.html que se genera con el método pushState() en los ejemplos anteriores se considera como una URL falsa, pero debería ser real. El propósito de esta API no es crear URL falsas sino proporcionar alternativas para los programadores con el fin de registrar la actividad del usuario en el historial para poder volver a un estado anterior en cualquier momento que se requiera (incluso después de que se cierre el navegador). Tendremos que asegurarnos de que el código en nuestro servidor devuelve este estado en el servidor y provee el contenido apropiado para cada URL solicitada (falsa o real).

18.1.5 Un ejemplo de la vida real

Lo que explicaremos a continuación constituye un ejemplo práctico de la vida real. Vamos a utilizar la API History y todos los métodos estudiados previamente para cargar cuatro imágenes desde el mismo documento. Cada imagen está asociada con una dirección URL falsa que se puede utilizar para retornar una imagen específica desde el servidor.

El documento principal se carga con una imagen por defecto. Esta imagen se asocia con el primero de cuatro enlaces que forman parte del contenido permanente del documento. Todos estos enlaces apuntan a direcciones URL falsas y referencian un estado y no un documento real, incluyendo el enlace para el documento principal, que se ha cambiado por pag1.html. Toda esta información cobra sentido más adelante.

Por ahora veamos el código de la plantilla HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
 <title>API History</title>
 <link rel="stylesheet" href="history.css">
 <script src="history.js"></script>
</head>
<body>
 <section id="paginaprincipal">
 Este contenido nunca es actualizado<br>
 <span id="url1">imagen 1</span> -
 <span id="url2">imagen 2</span> -
 <span id="url3">imagen 3</span> -
 <span id="url4">imagen 4</span> -
 </section>


 <aside id="datos">
 <img id="imagen" src="http://www.minkbooks.com/content/monster1.
 gif">
 </aside>
 </body>
 </html>
```

Código 18-5

Plantilla para una aplicación real.

La única diferencia entre esta nueva aplicación y la anterior es el número de enlaces y la cantidad de URL que se generan. En el contenido del Código 18-4 había dos estados: el estado 1 que correspondía al documento principal y el estado 2 que correspondía a la URL falsa (pag2.html) generada por el método pushState(). En este caso, hay que automatizar el proceso y generar un total de cuatro URL falsas correspondientes a cada imagen disponible.

```
function iniciar(){
 for(var f = 1; f < 5; f++){
  url = document.getElementById('url'+f);
  url.addEventListener('click', function(x){
   return function(){ cambiarpag(x);}
  }(f));
 }
 addEventListener('popstate', nuevaurl);
 history.replaceState(1, null, 'pag1.html');
}
function cambiarpag(pag){
 mostrar(pag);
 history.pushState(pag, null, 'pag'+pag+'.html');
}
function nuevaurl(e){
 mostrar(e.state);
}
function mostrar(current){
 if(current != null){
 image = document.getElementById('imagen');
 image.src = 'http://www.minkbooks.com/content/monster' + current +
  '.gif';
 }
}
addEventListener('load', iniciar)
```

Código 18-6

Cómo manipular el historial (history.js).

Como podemos observar, estamos utilizando las mismas funciones, pero con importantes cambios que están a la vista. En primer lugar, el método replaceState() en la función iniciar() tiene el atributo url que ha sido declarado como pag1.html. Decidimos programarlo así, declarando el estado del documento principal como 1 y la dirección URL como pag1.html (independientemente de la URL real del documento). De esta forma se hace más sencillo pasar de un estado a otro, de una URL a la otra, siempre con el mismo nombre y los mismos valores de la propiedad state para construir cada URL. Esto lo podemos constatar en la práctica a través de la función cambiarpag(). Cada vez que el usuario hace clic en uno de los enlaces de la plantilla se ejecuta esta función y la dirección URL falsa se construye con el valor de la variable pag y se añade dicha página al historial de la sesión. El valor recibido por esta función se declara previamente en el bucle for al principio de la función iniciar(). Este valor es declarado como 1 para el enlace página 1, 2 para el enlace página 2, y así sucesivamente.

Cada vez que se visita una URL, la función mostrar() es ejecutada para actualizar el contenido (la imagen) de acuerdo con dicha dirección URL. Debido a que el evento popstate a veces se activa cuando el valor de la propiedad state es null (como por ejemplo después de que el documento principal se carga por primera vez), comprobamos el valor recibido por la función mostrar() antes de hacer cualquier otra cosa. Si el valor difiere de null, esto significa que la propiedad state fue definida para esa URL y que la imagen correspondiente a ese estado es la que se muestra en pantalla.

Las imágenes que se utilizaron para este ejemplo fueron nombradas

monster1.gif, monster2.gif, monster3.gif y monster4.gif, siguiendo el mismo orden que los valores de la propiedad state. Por tanto, utilizando este valor, podemos seleccionar cuál imagen mostrar. Sin embargo, conviene recordar que los valores pueden ser los que se necesiten, y que el proceso para crear URL falsas y el contenido asociado a éstas tendrá que ir en consonancia con las necesidades de programación que se tengan.

Además, debemos tener en cuenta el hecho de que los usuarios deben ser capaces de volver a cualquiera de las URL generadas por la aplicación y ver el contenido que desean en la pantalla en cualquier momento. Debemos preparar el servidor para procesar estas URL con el fin de que cada estado (cada URL falsa) esté disponible y que sea siempre accesible. Por ejemplo, si un usuario abre una nueva ventana y escribe la URL page2.html en la barra de navegación, el servidor debe retomar el documento principal con la imagen monster2.gif y no solo la plantilla del Código 18-5. La idea principal que subyace detrás de esta API es proporcionar una alternativa a los usuarios de volver a cualquier estado anterior en cualquier momento que quieran, y solo podemos lograrlo convirtiendo las URL falsas en válidas.

Importante

El bucle for usado en el contenido del Código 18-6 para añadir un detector para el evento click a cada elemento en el documento utiliza una técnica de Javascript que sirve para enviar valores reales a una función. Para asignar un valor a la función que manejará el evento en un método addEventListener() se tiene que declarar el valor real. Si en su lugar enviamos una variable, lo que realmente estamos enviando no es el valor de la variable, sino una referencia a la misma. Así que en este caso, para enviar el valor actual de la variable f del bucle for se tienen que utilizar varias funciones anónimas. La primera función se ejecuta cuando se declara el método addEventListener(). Esta función recibe el valor actual de la variable f (debemos comprobar los paréntesis del final) y almacena este valor de la variable x. Entonces, la función retoma una segunda

función con el valor de la variable x. Esta segunda función es la que se ejecuta cuando se activa el evento en cuestión. Para obtener más información sobre este tema, visite nuestro sitio web y siga los enlaces correspondientes a este capítulo.

Hágalo usted mismo

Para probar el ejemplo anterior, utilice el documento HTML del Código 18-5 con los estilos CSS del Código 18-2. Copie el contenido del Código 18-6 en el archivo history.js y suba los archivos a su servidor. Abra el documento en su navegador y haga clic en cada uno de los enlaces. Navegue las URL que haya seleccionado mediante los botones de navegación. Las imágenes que aparecen en pantalla deben cambiar de acuerdo a la URL que aparezca en la barra de localización del navegador.

19 API Offline

19.1 El manifiesto

Los días en los que se trabajaba sin conexión a Internet han terminado. Como este capítulo se trata de la API Offline, la frase anterior puede parecer un poco extraña y sin sentido. Pero debemos pensar lo siguiente: hemos trabajado desconectados casi toda nuestra vida. Las principales herramientas que usábamos se encontraban entre las aplicaciones de escritorio. Ahora, de repente, la Red se ha convertido en la plataforma de trabajo. Las aplicaciones se han vuelto cada vez más complejas, y HTML5 está haciendo que la batalla entre las aplicaciones fuera de línea y las aplicaciones en línea sea cada vez más dura. Ahora tenemos a nuestra disposición bases de datos, acceso a archivos, almacenamiento, herramientas gráficas, de imagen y de edición de video, y multiprocesamiento, entre otras muchas facilidades que están disponibles en línea. Nuestra actividad diaria orienta su mirada hacia la Red, y nuestra productividad depende de ello. Los días de trabajar sin conexión han terminado definitivamente.

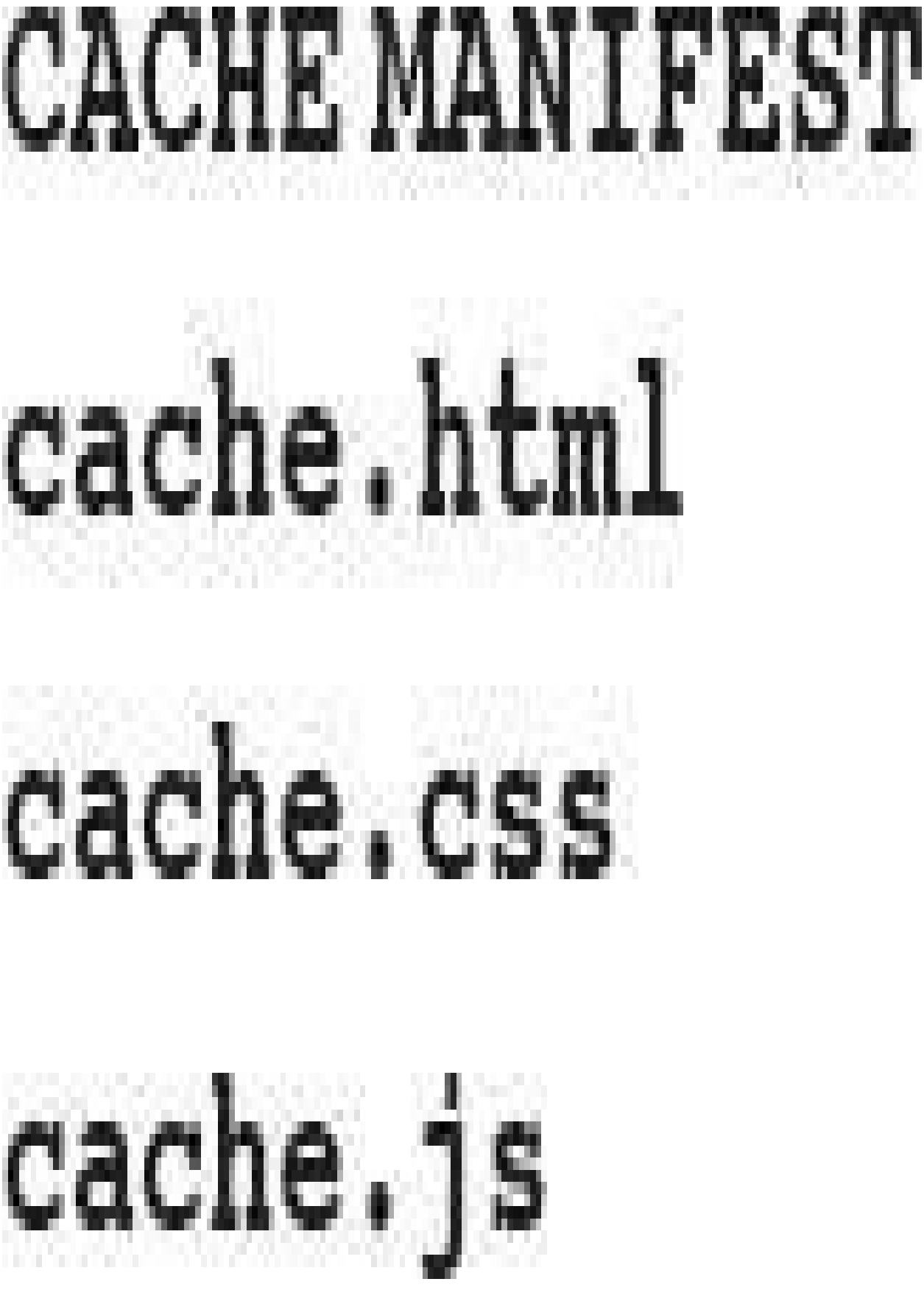
Sea como sea, la transición continúa y las aplicaciones web se vuelven cada vez más sofisticadas, exigen archivos más grandes y necesitan más tiempo para descargarse. Para cuando las aplicaciones en la Red hayan sustituido a las del escritorio, trabajar exclusivamente en línea será imposible. Los usuarios no podrán descargar archivos de gran tamaño cada vez que necesitan utilizar una aplicación o tener Internet disponible el 100% del tiempo. Las aplicaciones que no requieren Internet desaparecerán y las que funcionan solo en línea, tal como se usan en la actualidad, están destinadas al fracaso.

La API Offline está aquí para resolver este problema, ya que representa una alternativa para almacenar aplicaciones y archivos web en el ordenador del usuario para su uso futuro. Acceder a una aplicación una vez es suficiente para descargar todos los archivos que se necesitan para usar dicha aplicación cuando se está offline. Una vez que se han descargado los archivos, la aplicación

funciona en el navegador usando estos archivos, como lo haría una aplicación de escritorio, sin importar que se esté conectado o lo que pueda suceder con el servidor.

19.1.1 El archivo manifiesto

Tanto una aplicación web como un sofisticado sitio web consisten en una serie de archivos, no todos ellos imprescindibles para el funcionamiento de la aplicación. No todos estos archivos tienen que guardarse en el ordenador. La API asigna un archivo específico con una lista de aquellos archivos que son necesarios para trabajar fuera de línea. Este texto se denomina manifiesto (en inglés, manifest), y contiene una lista de las URL que contienen los archivos requeridos. El manifiesto se puede crear con cualquier editor de texto, salvarse con la extensión .appcache, y la primera línea del texto del contenido debe ser CACHE MANIFEST, tal como se muestra en el siguiente ejemplo.


CACHE MANIFEST
cache.html
cache.css
cache.js

Código 19-1

Creación del archivo CACHE MANIFEST.

Los archivos que se mencionan en el CACHE MANIFEST constituyen todos los archivos y aplicaciones que se necesitan para trabajar en el ordenador del usuario sin requerir ningún recurso extra.

En el ejemplo que se muestra en el Código 19-1 tenemos el archivo cache.html como documento principal, el archivo cache.css con los estilos CSS y el archivo cache.js para los códigos de Javascript.

19.1.2 Categorías

Así como especificamos los archivos que se necesitan para que una aplicación funcione sin conexión, de ese mismo modo necesitamos puntualizar acerca de los archivos necesarios que deben estar disponibles solo en línea. Esto se aplica por ejemplo en el caso de aplicaciones que solo funcionan si estamos conectados, como es el caso de una sala de chat para efectuar consultas.

Para poder identificar estos archivos en el archivo manifiesto, en API contamos con tres categorías:

CACHE: Representa la categoría por defecto y todos los archivos dentro de esta categoría se guardan en el ordenador del usuario para su uso futuro.

NETWORK: Se considera una lista de aprobación (que no es spam) y todos los archivos de esta clasificación solo están disponibles online.

FALLBACK: Esta clasificación se utiliza para los archivos que pueden ser útiles cuando se obtienen del servidor estando online, pero que deben ser reemplazados por su version offline. Si el navegador detecta esta conexión, tratará de usar la version original. Si no ocurre así, será usado el que se ubique en el ordenador del usuario.

Si aplicamos las tres categorías, el archivo manifiesto resultará algo como lo que sigue:

```
CACHE MANIFEST

CACHE:
cache.html
cache.css
cache.js

NETWORK:
chat.html

FALLBACK:
noticias.html sinnoticias.html
```

Código 19-2

Declaración de archivos por categoría.

En este nuevo archivo manifiesto que se muestra en el Código 19-2, los archivos son listados cada uno en la categoría que le corresponde.

Los tres archivos que se encuentran en la categoría CACHE se descargan, se almacenan en el ordenador del usuario y se utilizan en la aplicación desde ese momento (a menos que se especifique algo diferente después). El archivo chat.html que se especifica en la categoría NETWORK estará disponible solo si el navegador está conectado a Internet. Por otro lado, el archivo newslist.html, que pertenece a la categoría FALLBACK, será utilizado cuando el usuario esté online y, si no se puede conectar, será reemplazado automáticamente por el archivo nonews.html. Así ocurre también con los archivos de la categoría CACHE, el archivo nonews.html estará en el caché y quedará guardado en el ordenador del usuario para cuando sea necesario utilizarlo.

La categoría FALLBACK no solo es útil para reemplazar archivos individuales, sino que también lo es para sustituir directorios completos. Por ejemplo, la línea / noconnection.html reemplazará a cualquier archivo que no esté en el caché con el archivo noconnection.html. Ésta es una buena forma de desviar al usuario hacia un documento que recomiende estar online cuando intente acceder a una parte de la aplicación que no está disponible fuera de línea.

19.1.3 Comentarios

Los comentarios se pueden añadir en un archivo manifiesto utilizando el símbolo almohadilla (#), uno por cada línea. Ya que los archivos están ordenados por categorías, los comentarios pueden parecer menos importantes, pero no lo son, ya que resultan útiles para las actualizaciones en el caché.

El archivo manifiesto no solo evidencia lo que está en el caché, sino el momento en que debe estar almacenado allí. Cada vez que se actualizan los archivos de la aplicación, la única forma de que lo procesen los navegadores es a través del archivo manifiesto. Si los archivos actualizados son iguales a los anteriores, no se añade ninguno a la lista y el archivo manifiesto se queda como estaba, de manera que el navegador no podrá reconocer la diferencia y continuará utilizando los archivos antiguos que tiene en el caché.

De todas formas, para forzar al navegador a descargar de nuevo los archivos, lo que podemos hacer es indicar dicha actualización a través de los comentarios. Será suficiente un comentario con la fecha de la última actualización, tal como se muestra en el siguiente ejemplo:

```
CACHE MANIFEST

CACHE:
cache.html
cache.css
cache.js

NETWORK:
chat.html

FALLBACK:
/ sinconexión.html

# fecha 2013/05/10
```

Código 19-3

Uso de comentarios para informar sobre las actualizaciones.

Supongamos que añadimos más códigos a las funciones del archivo cache.js. Los usuarios ya tendrán el archivo en el caché de sus ordenadores y los navegadores usarán esta versión antigua en lugar de la nueva. Cambiando la fecha al final del archivo manifiesto, o añadiendo nuevos comentarios para informar a los navegadores sobre las actualizaciones, se logra que los archivos sean descargados de nuevo, incluyendo la versión más actual del archivo cache.js y así se podrá trabajar sin conexión.

Una vez actualizado el caché, el navegador usará nuevas copias de los archivos para que la aplicación pueda funcionar.

19.1.4 Uso del archivo manifiesto

Luego de seleccionar todos los archivos necesarios para que la aplicación funcione fuera de línea y de preparar una lista de URL que apunten a estos archivos, debemos incluir el archivo manifiesto en nuestros documentos. HTML5 proporciona un nuevo atributo <html> para indicar la ubicación del archivo.

```
<!DOCTYPE html>
<html lang="es" manifest="micache.appcache">

<head>
 <title>API Offline</title>
 <link rel="stylesheet" href="cache.css">
 <script src="cache.js"></script>
</head>
<body>
 <section id="caja">
 Aplicación para trabajar fuera de línea
 </section>
</body>
</html>
```

Código 19-4

Carga del archivo manifiesto.

El Código 19-4 muestra un documento HTML que incluye el atributo manifest en el elemento <html>. El atributo manifest indica la localización del archivo con el mismo nombre, imprescindible para generar el caché de la aplicación.

Como podrá ver, no cambia nada en el resto del documento: los archivos para estilos CSS y los códigos de Javascript se incluyen como de costumbre, independientemente del contenido del archivo manifiesto.

El archivo CSS solo debe incluir los estilos para el apartado <section> de nuestro documento. Puede crear sus propios estilos o utilizar los siguientes:

```
#caja{
 width: 500px;
 height: 300px;
 margin: 10px;
 padding: 10px;
 border: 1px solid #999999;
}
```

Código 19-5

Regla CSS para datos.

El archivo manifiesto debe salvarse con la extensión .appcache y darle el nombre que se desee (para este ejemplo hemos usado micache). Cada vez que el navegador encuentre el atributo manifest en un documento, intentará descargar el archivo manifiesto en primer lugar y luego todos los archivos listados en su interior. El atributo manifest debe ser incluido en cada documento HTML que tiene que ser parte del caché de la aplicación. El proceso es transparente para el usuario y puede ser controlado desde código Javascript usando la API, como se verá más adelante.

Importante

El tipo MIME text/cache-manifest no forma parte de la configuración de ningún servidor. Debe añadirse manualmente. La forma de incluir este nuevo archivo dependerá del tipo de servidor que se tenga. Para algunas versiones de Apache, por ejemplo, será suficiente añadir la siguiente línea en el archivo httpd.conf para disponer del tipo MIME indicado: AddType text/cache-manifest.appcache.

Aparte de la extensión y de la estructura interna del archivo manifiesto, hay otro requisito importante a tener en cuenta. El archivo manifiesto debe proveerlo el servidor con el tipo MIME adecuado. A cada archivo le corresponde un tipo MIME que indica el formato de su contenido. Por ejemplo, el tipo MIME para un archivo HTML es text/html. El archivo manifiesto debe ser provisto usando el

tipo MIME text/cache-manifest o de lo contrario devolverá un error.

Una manera fácil de hacerlo es añadiendo una línea adicional en el archivo .htaccess. Muchos servidores disponen de este archivo de configuración en la carpeta raíz de cada sitio web. La sintaxis sería: Addtype MIME/tipo extensión (por ejemplo, AddType text/cache-manifest .appcache).

19.2 API Offline

El archivo manifiesto por sí solo debería bastar para generar un caché que funcione para sitios web pequeños o códigos simples, pero las aplicaciones complejas requieren mayor control. El archivo manifiesto declara los archivos necesarios para el caché, pero no puede informar sobre cuántos de estos archivos ya fueron descargados, o los errores encontrados en el proceso, o cuándo hay una actualización lista para ser utilizada, entre otras cosas. Considerando estos posibles escenarios, la API provee el nuevo objeto ApplicationCache con métodos, propiedades y eventos para controlar todo el proceso.

19.2.1 Los errores

Probablemente el evento más importante del mencionado objeto ApplicationCache es error. Si el mismo ocurre durante el proceso de lectura de archivo del servidor, no podrá ser creado o actualizado el caché necesario para que la aplicación trabaje sin conexión. Es sumamente importante reconocer estas situaciones y actuar de acuerdo a las circunstancias. Usando el documento HTML presentado en el Código 16-4, vamos a construir una pequeña aplicación para demostrar cómo funciona este evento.

```
function iniciar(){
 var cache = applicationCache;
 cache.addEventListener('error', mostrarerror);
}
function mostrarerror(){
 alert('error');
}
addEventListener('load', iniciar);
```

Código 19-6

Control de los errores.

El atributo applicationCache usado en el Código 19-6 retorna el objeto ApplicationCache para este documento. Después de almacenar una referencia para el objeto dentro de la variable cache, se añade un detector para el evento error. Este detector llamará a la función showerror() y se desplegará un mensaje de alerta en pantalla informando acerca de dicho error.

Hágalo usted mismo

Genere un archivo HTML con el Código 19-4, un archivo Javascript llamado cache.js con el contenido del Código 19-6 y un archivo manifiesto llamado mycache.appcache. Como vimos anteriormente, debemos incluir un listado de archivos en el caché dentro de la categoría CACHE en el archivo manifiesto. Para nuestro ejemplo, estos archivos son el archivo HTML y el archivo cache.js y el archivo cache.css con los estilos que les correspondan. Suba estos archivos a su servidor. Si por casualidad borra el archivo manifiesto u olvida añadir el tipo MIME correspondiente a este archivo en su servidor, el evento error se disparará. También puede interrumpir el acceso a la Red o usar la opción Trabajar sin conexión de Mozilla Firefox para ver la aplicación desde el nuevo caché y seguir trabajando desconectado.

19.2.2 online y offline

Una nueva propiedad fue incorporada para el objeto Navigator. Se trata de onLine e indica el estado actual de la conexión. Esta propiedad tiene dos eventos asociados que serán disparados cuando su valor cambie. La propiedad y los eventos no son parte del objeto ApplicationCache, y son útiles para esta API.

online: Este evento es disparado cuando el valor de la propiedad onLine cambia a true (verdadero).

offline: Este evento es disparado cuando el valor de la propiedad onLine cambia a false (falso).

He aquí un ejemplo de cómo utilizar las propiedades descritas:

```
var caja;
function iniciar(){
 caja = document.getElementById('caja');

 addEventListener('online', function(){ estado(1); });
 addEventListener('offline', function(){estado (2); });
}

function estado (valor){
 switch(valor){
 case 1:
 caja.innerHTML += '<br>Está en línea';
 break;
 case 2:
 caja.innerHTML += '<br>Está fuera de línea';
 break;
 }
}
addEventListener('load', iniciar);
```

Código 19-7

Verificación del estado de la conexión usando la propiedad online.

En el contenido del Código 19-7 utilizamos funciones anónimas para manejar los eventos y enviar un valor a la función estado() para mostrar el mensaje que corresponde en datos. Los eventos se dispararán cada vez que cambia la propiedad onLine.

Hágalo usted mismo

Use los mismos archivos HTML y CSS de los ejemplos previos. Copie el contenido del Código 19-7 en el archivo cache.js. Usando Mozilla Firefox, elimine el caché de su sitio web y abra el documento HTML. Para probar los eventos use la opción Trabajar sin conexión de Mozilla Firefox. Cada vez que haga clic en esta opción, cambian las condiciones y se agrega un nuevo mensaje en datos.

Importante

No hay garantía de que la propiedad vuelva a tener siempre el valor adecuado. Experimentar estos eventos en un ordenador de escritorio probablemente no producirá ningún efecto, aun cuando esté totalmente desconectado de Internet.

Para probar este ejemplo en un PC, recomendamos usar la opción Trabajar sin conexión de Mozilla Firefox.

19.2.3 Estado del caché

El hecho de crear o de actualizar el caché puede llevar desde unos pocos segundos hasta varios minutos, según el tamaño de los archivos a descargar. El proceso completo pasa por diferentes fases de acuerdo con lo que el servidor pueda hacer en cada momento. En una actualización promedio, por ejemplo, los navegadores intentarán leer el archivo manifiesto para verificar las actualizaciones e informar cuando se hayan finalizado las mismas. Para informar sobre cada paso del proceso, la API ofrece la propiedad status. Esta propiedad puede tomar los valores siguientes:

UNCACHED (valor 0): Este valor indica que no se ha creado ningún caché para esta aplicación.

IDLE (valor 1): Este valor indica que el caché de la aplicación es el más nuevo y que no ha quedado obsoleto.

CHECKING (valor 2): Este valor indica que el navegador está buscando nuevas actualizaciones.

DOWNLOADING (valor 3): Este valor indica que los archivos del caché se están descargando.

UPDATEREADY (valor 4): Este valor indica que el caché de la aplicación está disponible y que no es obsoleto, pero que no es el más nuevo y que hay una actualización lista para reemplazarlo.

OBSOLETE (valor 5): Este valor indica que el caché actual está obsoleto.

Se puede revisar el valor de la propiedad status en cualquier momento, pero lo mejor es utilizar los eventos provistos por el objeto ApplicationCache para controlar el estado del proceso y el propio caché. Los eventos que se explican a continuación se disparan formando una secuencia, y algunos de ellos se asocian con una aplicación específica del estado del caché:

checking: Este evento se dispara cuando el navegador está controlando si hay actualizaciones.

noupdate: Este evento se dispara cuando no se encuentran cambios en el archivo manifiesto.

downloading: Este evento se dispara cuando el navegador encuentra una nueva actualización y comienza a descargar los archivos.

cached: Este evento se dispara cuando está listo el caché.

updateready: Este evento se dispara cuando el proceso de descarga para una actaualización se ha completado.

obsolete: Este evento se dispara cuando el archivo manifiesto ya no está disponible y el caché está siendo eliminado.

El siguiente ejemplo nos ayuda a entender el proceso. A través del código, cada vez que se dispara un evento, se añade un mensaje a datos con el valor del evento y de la propiedad status.

```
var cache, caja;
function iniciar(){
  caja = document.getElementById('caja');
  cache = applicationCache;

  cache.addEventListener('checking', function(){ mostrar(1); });
  cache.addEventListener('downloading', function(){ mostrar(2); });
  cache.addEventListener('cached', function(){ mostrar(3); });
  cache.addEventListener('updateready', function(){ mostrar(4); });
  cache.addEventListener('obsolete', function(){ mostrar(5); });
}
function mostrar(valor){
 caja.innerHTML += '<br>Mi estado: '+ cache.status;
 caja.innerHTML += ' | Evento: '+ valor;
}
addEventListener('load', iniciar);
```

Código 19-8

Control de la conexión.

Usamos una función anónima para responder a los eventos y así enviar un valor que permita identificar cada evento luego en la función mostrar(). Este valor y el valor de la propiedad status se muestran en pantalla según el estado del caché cada vez que se carga el documento.

Hágalo usted mismo

Use los archivos HTML y los estilos CSS de los ejemplos anteriores. Copie el contenido del Código 19-8 en el archivo cache.js. Suba la aplicación al servidor, y verá cómo los diferentes pasos del proceso se muestran en pantalla de acuerdo al estado del caché cada vez que se carga el documento.

Importante

Si el caché ya ha sido creado, es fundamental seguir los pasos para limpiar el caché antiguo y cargar la versión nueva. Un paso es modificar el archivo manifiesto, pero no es el único. Los navegadores conservan una copia de los archivos durante algunas horas antes de considerar siquiera comprobar si hay actualizaciones, por lo cual no importa cuántos comentarios nuevos nuevos haya

o cuántos archivos se añadan al archivo manifiesto, igualmente el navegador utilizará la versión del caché por un tiempo. Para hacer una prueba de lo que hemos explicado, recomendamos cambiar los nombres de cada archivo. De hecho, al añadir un número al final del nombre del archivo (por ejemplo cache2.js) hará que el navegador lo considere como una nueva aplicación y que genere un nuevo caché. Esto solo es útil como una prueba con fines didácticos.

19.2.4 Progreso

Las aplicaciones pueden incluir imágenes, archivos de códigos, información para bases de datos, vídeos o cualquier otro archivo grande que pueda requerir mucho tiempo para descargarse. Para monitorizar este proceso, la API provee el conocido evento progress. Este evento es el mismo que usamos en capítulos anteriores.

El evento progress solo se dispara mientras los archivos se están descargando. En el ejemplo que viene a continuación vamos a utilizar el evento noupdate junto con los eventos cached y updateready para informar acerca de cuándo ha terminado el proceso.

```
var cajadatos;
function iniciar(){
  cajadatos = document.getElementById('cajadatos');
  cajadatos.innerHTML = '<progress value="0" max="100">0%</
 progress>';
  var cache = applicationCache;
  cache.addEventListener('progress', progreso);
  cache.addEventListener('cached', mostrar);
  cache.addEventListener('updateready', mostrar);
  cache.addEventListener('noupdate', mostrar);
}

function progreso(e){
  if(e.lengthComputable){
    var per = parseInt(e.loaded / e.total * 100);
    var barraprogreso = cajadatos.querySelector("progress");
    barraprogreso.value = per;
    barraprogreso.innerHTML = per + '%';
  }
}
function mostrar(){
  cajadatos.innerHTML = 'Hecho';
}
addEventListener('load', iniciar);
```

Código 19-9

Progreso de la descarga.

Como de costumbre, el evento progress se dispara periódicamente para informar acerca del estado del proceso. En el contenido del Código 19-9 cada vez que se dispara el evento progress la función progreso() es llamada y la información en pantalla es actualizada usando un elemento <progress>.

Existen diferentes posibilidades al final del proceso. La aplicación podría haber sido almacenada en el caché por primera vez, con lo cual se dispararía el evento cached. Puede ocurrir que el caché ya exista y que esté disponible una actualización, por lo que cuando los archivos se descarguen finalmente se dispare el evento updateready. Y una tercera posibilidad es que el caché se encuentre en uso y que no esté disponible ninguna actualización, por lo que se dispara el evento noupdate. Tomamos en cuenta los eventos para cada uno de los casos y llamamos a la función mostrar() en todos ellos para imprimir el mensaje Hecho en la pantalla, indicando de esta forma que se ha completado el proceso. Más adelante en este libro se aporta una explicación más detallada del evento progress y sus propiedades.

Hágalo usted mismo

Use el archivo HTML y los estilos CSS de los ejemplos anteriores. Copie el contenido del Código 19-8 en el archivo cache.js. Suba la aplicación a su servidor y cargue el documento principal. Debe incluir un archivo de gran tamaño para poder ver la barra de progreso trabajando (en la actualidad los

navegadores tienen limitaciones en el tamaño del caché. Recomendamos probar con algunos archivos de no más de cinco megabytes). Por ejemplo, al usar el vídeo trailer.ogg que se introdujo ya en este libro, el archivo manifiesto debe lucir de la siguiente forma:

```
CACHE MANIFEST

cache.html

cache.css

cache.js

trailer.ogg


# date 2013/01/03
```

Importante

Utilizamos innerHTML para añadir un nuevo elemento <progress> al documento. Esta no es una práctica recomendada, pero es útil para lo que queremos explicar y ejemplificar. Normalmente los elementos se agregan al documento usando el método Javascript createElement() junto con appendChild(), tal como se explica en el Capítulo 4.

19.2.5 Actualización del caché

Hasta aquí hemos visto cómo crear un caché para nuestra aplicación, cómo informar al navegador cuando la actualización está disponible y cómo controlar el proceso cada vez que un usuario accede a la aplicación. Esto es muy útil pero no del todo transparente para el usuario. El caché y sus actualizaciones se cargan tan pronto como el usuario ejecuta la aplicación, lo que puede producir retrasos y mal funcionamiento. La API resuelve este problema incorporando nuevos métodos para actualizar el caché mientras la aplicación está en marcha:

update(): este método inicia una actualización del caché. Indica al navegador que descague en primer orden el archivo manifiesto y que continúe con el resto de archivos si detecta un cambio en el archivo manifiesto (si los archivos del caché se hubiesen modificado).

swapCache(): este método activa el caché más reciente después de una actualización. No ejecuta un nuevo código ni reemplaza recursos, pero indica al navegador que un nuevo caché está disponible para ser leído.

Para actualizar el caché, lo que se necesita es llamar al método update(). Los eventos updateready y noupdate son útiles para saber el resultado del proceso. En el siguiente ejemplo vamos a usar un nuevo documento HTML con dos botones que se usan para solicitar la actualización y comprobar cuál es el código que se encuentra actualmente en el caché.

```
<!DOCTYPE html>
<html lang="en" manifest="micache.appcache">
<head>
 <title>API Offline</title>
 <link rel="stylesheet" href="cache.css">
 <script src="cache.js"></script>
</head>
<body>
 <section id="caja">
   Aplicación Offline
 </section>
 <input type="button" id="update" value="Cargar caché">
 <input type="button" id="test" value="Prueba">
</body>
</html>
```

Código 19-10

Documento HTML para probar el método update().

El código Javascript implementa técnicas que ya hemos estudiado, solo tenemos que incluir dos nuevas funciones para los botones:

```
var cache, caja;
function iniciar(){
 caja = document.getElementById('caja');

 var update = document.getElementById('update');
 update.addEventListener('click', updatecache);
 var test = document.getElementById('test');
 test.addEventListener('click', testcache);

 cache = applicationCache;
 cache.addEventListener('updateready', function(){ mostrar(1); });
 cache.addEventListener('noupdate', function(){ mostrar(2); });
}
function updatecache(){
 cache.update();
}
function testcache(){
 caja.innerHTML += '<br>cambiar este mensaje';
}
function mostrar(value){
 switch(value){
 case 1:
 caja.innerHTML += '<br>Actualización lista';
 break;
 case 2:
 caja.innerHTML += '<br>No hay actualizaciones disponibles';
 break;
 }
}
addEventListener('load', iniciar);
```

Código 19-11

Actualización del caché y comprobación de la versión actual.

En la función iniciar() se agrega un evento click para los dos botones. Haciendo clic en el botón Cargar caché se llama a la función updatecache() y se ejecuta el método update(), y haciendo clic en el botón Prueba se llamará a la función testcache() y se muestra un texto en la caja datos. Este texto puede modificarse luego para crear una nueva versión del código y verificar si ha sido actualizado o no.

Hágalo usted mismo

Cree un nuevo documento HTML con el contenido del Código 19-10. El archivo manifiesto y los estilos CSS son los mismos que mostramos en los ejemplos anteriores (a menos que usted haya cambiado algunos nombres de archivo, en cuyo caso tendrá que actualizar la lista de archivos dentro del archivo manifiesto). Copie el contenido del Código 19-11 en un archivo llamado cache.js, y suba todos estos archivos a su servidor. Abra el documento principal en su navegador y pruebe la aplicación.

Una vez que se carga el documento HTML, la ventana muestra la caja datos y dos botones debajo. Tal como explicamos anteriormente, el botón Actualizar caché tiene asociado el evento click con la función updatecache(). Si se hace clic en ese botón, el método update() se ejecutará dentro de esta función y el proceso de actualización comenzará. El navegador descargará el archivo manifiesto y lo

comparará con el archivo que estaba previamente en el caché. Si el archivo se ha modificado, todos los archivos listados dentro serán descargados de nuevo. Cuando el proceso ha terminado, el evento updateready se dispara. Este evento llama a la función mostrar() con valor 1, que corresponde al mensaje Actualización lista. Por otro lado, si el archivo manifiesto no cambia, no es detectada ninguna actualización y no se dispara ningún evento noupdate. Este evento llama a la función mostrar() con el valor 2. En este caso, se muestra el mensaje No hay actualizaciones disponibles en la caja datos.

Usted puede comprobar cómo funciona el código modificando o añadiendo comentarios al archivo manifiesto. Cada vez que se presiona el botón para actualizar el caché luego de una modificación, aparecerá el mensaje Actualización lista en datos. También puede probar a cambiar el texto en la función testcache() para detectar si ya se ha puesto en marcha una actualización del caché.

Importante

En esta oportunidad no hay necesidad de borrar el caché del navegador para descargar la nueva versión. El método update() obliga al navegador a descargar el archivo manifiesto y el resto de archivos si detecta la actualización. De todas maneras, el nuevo caché no estará disponible sino hasta que el usuario reinicie la aplicación.

20 API Page Visibility

20.1 El estado de visibilidad

Las aplicaciones Web son cada vez más sofisticadas y ahora exigen más recursos informáticos que nunca antes. Las páginas web ya no son documentos estáticos; Javascript las ha convertido en aplicaciones completas, capaces de ejecutar procesos complejos sin interrupción e incluso sin la intervención del usuario. Pero hay momentos en que estos procesos pueden ser cancelados o detenidos momentáneamente para utilizar con eficacia los recursos y también para proporcionar una mejor experiencia de usuario. Con la intención de producir aplicaciones conscientes, HTML5 introduce la API Page Visibility. Esta API informa a la aplicación sobre el estado de visibilidad actual del documento, comunicando cuando la pestaña está oculta o se minimiza la ventana, para que pueda decidir qué hacer mientras nadie está mirando la página.

Importante

Para el momento de escribir este manual, el evento y la propiedad necesitan un prefijo para ser interpretados por los navegadores. Google Chrome utiliza webkit-visibilitychange y webkit-Visibili-tyState mientras Mozilla Firefox utiliza mozvisibility-change y mozVisibilityState. Los ejemplos de este capítulo se prepararon para Google Chrome.

20.1.1 Estado actual

Básicamente, la API proporciona una propiedad para informar sobre el estado actual y un evento para permitir que la aplicación sepa que algo ha cambiado.

visibilityState: Esta propiedad devuelve el estado de visibilidad actual del documento. Los valores posibles son hidden o visible (también pueden ser aplicados por los navegadores algunos valores opcionales tales como prerender y unloaded).

visibilitychange: Este evento se activa cuando el valor de la propiedad visibilityState cambia.

```
<!DOCTYPE html>
<html lang="en">
<head>
 <title>API Page Visibility</title>
 <script>
  function iniciar(){
   document.addEventListener('webkitvisibilitychange', mostrar);
  }
  function mostrar(){
   var elemento = document.getElementById('application');
   elemento.innerHTML += '<br>' + document.webkitVisibilityState;
  }
 addEventListener('load', iniciar);
 </script>
</head>
<body>
 <section id="application">
 Cambia a otra pestaña o minimiza esta ventana para cambiar la
 visibilidad de la página.
 </section>
</body>
</html>
```

Código 20-1

Informar sobre el estado de visibilidad.

En el Código 20-1, la función mostrar() está establecida para manejar el evento visibilitychange. Cuando el evento se activa, la función muestra el valor de la propiedad visibilityState en la pantalla para informar acerca del estado actual del documento. Cuando la pestaña es sustituida por otra pestaña o la ventana es minimizada, el valor de visibilityState cambia a hidden, y cuando la pestaña o ventana se restaura, el valor se establece de nuevo como visible.

20.1.2 Una mejor experiencia

Una de las razones para la aplicación de esta API era el elevado consumo de recursos por parte de las aplicaciones modernas, pero la API también fue diseñada como una forma de mejorar la experiencia del usuario. El siguiente ejemplo le muestra cómo lograr esto último con solo unas pocas líneas de código.

```
<!DOCTYPE html>
<html lang="es">
<head>
 <title>API Page Visibility</title>
 <script>
  var video;
  function iniciar(){
   video = document.getElementById('media');
   document.addEventListener('webkitvisibilitychange', mostrar);
   video.play();
  }
  function mostrar(){
   var estado = document.webkitVisibilityState;
   switch(estado){
    case 'visible':
    video.play();
    break;
   case 'hidden':
    video.pause();
    break;
   }
 }
 addEventListener('load', iniciar);
 </script>
</head>
<body>
 <video id="media" width="720" height="400">
  <source src="http://minkbooks.com/content/trailer.mp4">
  <source src="http://minkbooks.com/content/trailer.ogg">
 </video>
</body>
</html>
```

Código 20-2

En respuesta al estado de visibilidad.

El Código 20-2 reproduce un vídeo mientras que el documento está visible y se detiene cuando no lo está. Usa las mismas funciones del ejemplo anterior, excepto que esta vez el valor de la propiedad visibilityState es comprobado por una declaración switch (cambiar) y los métodos play() o pause()se ejecutan de acuerdo con la situación.

Hágalo usted mismo

Cree un archivo HTML con el Código 20-2. Abra el archivo en el navegador y alterne entre las pestañas para ver la aplicación en acción. El vídeo se detendrá cuando el documento no sea visible y se reanudará cuando sea visible de nuevo.

20.1.3 Detector completo

Por alguna razón, los navegadores no cambian el valor de la propiedad visibilityState cuando el usuario abre una nueva ventana del navegador o programa. La API solo es capaz de detectar el cambio en la visibilidad cuando la pestaña está oculta por otra pestaña o está en una ventana minimizada. Para determinar el estado de visibilidad en cualquier circunstancia, podemos complementar la API con los eventos blur y focus. Estos son eventos tradicionales de Javascript disparados cuando la ventana o cualquier elemento del documento pierde o gana enfoque, respectivamente. Mediante la adición de una pequeña función se pueden combinar todas las herramientas disponibles para construir un mejor detector.

```
<!DOCTYPE html>
<html lang="es">
<head>
 <title>API Page Visibility</title>
 <script>
  var estado;
  function iniciar(){
   addEventListener('blur', function(){ cambiar('hidden'); });
   addEventListener('focus', function(){ cambiar('visible'); });
   document.addEventListener('webkitvisibilitychange', function(){
              cambiar(document.webkitVisibilityState); });
 }
 function cambiar(nuevoestado){
  if(estado != nuevoestado){
   estado = nuevoestado;
   mostrar();
 }
 }

 function mostrar(){
  var elemento = document.getElementById('application');
  elemento.innerHTML += '<br>' + estado;
 }


  addEventListener('load', iniciar);
 </script>
</head>
<body>
 <section id="application">
  Move to another window to change the visibility state
 </section>
</body>
</html>
```

Código 20-3

La combinación de los eventos blur, focus y visibilitychange.

En la función iniciar() del Código 20-3 hemos añadido detectores para los tres eventos. La función cambiar() se ajusta para controlar el evento y procesar el valor que corresponde al estado actual. Este valor es determinado por cada evento. El evento blur envía el valor hidden, el evento enfoque envía el valor visible, y el evento visibilitychange envía el valor actual de la propiedad visibilityState. Como resultado, la función cambiar() recibe el valor correcto, no importa si el documento está oculto (en otra pestaña, ventana o programa). Mediante el uso de la variable estado, la función compara el valor anterior con el nuevo, almacena el nuevo valor en la variable y llama a la función mostrar() para mostrar el estado en la pantalla.

Este proceso es un poco más complicado que el anterior, pero considera todas las situaciones posibles en las que el documento puede estar oculto o cambiar de estado.

Importante

No está claro por el momento si la API debe funcionar como tal o si las funciones desempeñadas por los eventos blur y focus más tarde serán añadidas a la especificación. Estos eventos no son totalmente confiables, pero la API parece centrarse solo en las pestañas, haciendo caso omiso de las ventanas. Si las premisas de HTML5 se mantienen, esta situación debería ser corregida pronto. Para obtener información actualizada, visite www.minkbooks.com/updates/.

21 Ajax Level 2

21.1 XMLHttpRequest

El viejo paradigma de la Red exigía que los sitios web y aplicaciones accedieran al servidor y proporcionaran toda la información al mismo tiempo. Si se solicitaba nueva información, el navegador tenía que acceder al servidor y sustituir la información anterior por la nueva. Esto derivó en la utilización la palabra “páginas” para los documentos HTML. Los documentos se sustituían unos a otros como se pasan las páginas de un libro.

Esta fue la norma hasta que alguien encontró un mejor uso para un viejo objeto, introducido por primera vez por Microsoft y posteriormente mejorado por Mozilla, llamado XMLHttpRequest Este objeto proporciona una manera de acceder al servidor y recuperar información de Javascript sin volver a cargar el documento HTML. En un artículo escrito en 2005, fue acuñado el nombre de “Ajax” para este procedimiento.

Debido a su importancia en las aplicaciones modernas, HTML5 ha introducido la nueva API XMLHttpRequest Level 2 para realizar Ajax. Esta nueva API incorpora características como la comunicación entre orígenes y nuevos eventos para controlar la evolución de la solicitud. Estas mejoras simplifican los códigos y proporcionan nuevas opciones, como la interacción con varios servidores desde la misma aplicación o el trabajo con pequeños fragmentos de datos en lugar de archivos enteros, por nombrar algunas.

El elemento más importante de la API Ajax Level 2 es, por supuesto, el objeto XMLHttpRequest, que se crea a través de un constructor:

XMLHttpRequest(): Este constructor devuelve un objeto XMLHttpRequest desde el que podemos empezar una petición y escuchar los eventos para controlar el proceso de comunicación.

El objeto creado por el constructor XMLHttpRequest() tiene métodos importantes para iniciar y controlar la solicitud:

open(método, URL, asinc): Este método configura una solicitud pendiente. El atributo método especifica el método HTTP usado para abrir la conexión (GET o POST), el atributo url declara la ubicación de la secuencia de comandos que va a procesar la solicitud y asinc es un valor booleano que establece comunicación síncrona (false) o asíncrona (true). El método también puede incluir valores para el usuario y una contraseña cuando es necesario.

send(datos): Éste es el método que realmente inicia la petición. En un objeto XMLHttpRequest hay varias versiones de este método para procesar diferentes tipos de datos. El atributo datos puede ser omitido, o puede ser declarado como un ArrayBuffer, una nota, un documento, una cadena o un objeto FormData.

abort(): Éste es un método sencillo que permite cancelar la solicitud.

21.1.1 Recuperar datos

Vamos a comenzar a construir un ejemplo para obtener información de un archivo de texto en el servidor mediante el método GET. Para hacerlo vamos a necesitar un nuevo documento HTML con un botón para iniciar la solicitud:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title> Ajax Level 2 </title>
  <link rel="stylesheet" href="ajax.css">
  <script src="ajax.js"></script>
</head>
<body>
  <section id="cajaform">
   <form name="form">
    <input type="button" id="boton" value="Adelante">
   </form>
  </section>
 <section id="cajadatos"></section>
</body>
</html>
```

Código 21-1

Documento HTML para peticiones Ajax.

Para que los códigos resulten tan sencillos como sea posible, mantenemos nuestra estructura HTML usual y aplicaremos algunos estilos básicos con fines estéticos.

```
#cajaform{
 float: left;
 padding: 20px;
 border: 1px solid # 999999;
}
#cajadatos{
 float: left;
 ancho: 500px;
 margin-left: 20px;
 padding: 20px;
 border: 1px solid # 999999;
}
```

Código 21-2

Estilos para las cajas.

Hágalo usted mismo

Cree un archivo HTML con el Código 21-1 y un archivo CSS llamado ajax.css con las reglas del Código 21-2. Para poder probar los ejemplos siguientes, tendrá que subir todos los archivos, incluido el archivo de Javascript y el archivo al que se presente la solicitud, a su servidor. Proporcionaremos nuevas instrucciones en cada ejemplo.

El siguiente script lee un archivo del servidor y muestra su contenido en la pantalla. No se envía ningún dato al servidor; solo tenemos que hacer una solicitud GET y mostrar la información obtenida.

```
var cajadatos;
function iniciar(){
 cajadatos = document.getElementById('cajadatos');
 var boton = document.getElementById('boton');
 boton.addEventListener('click', leer);
}
function leer(){
 var url = "archivodetexto.txt";
 var solicitud = new XMLHttpRequest();
 solicitud.addEventListener('load', mostrar);
 solicitud.open("GET", url, true);
 solicitud.send(null);
}
 function mostrar(e){
 var datos = e.target;
 if(datos.status == 200){
 cajadatos.innerHTML = datos.responseText;
 }
}
addEventListener('load', iniciar);
```

Código 21-3

Lectura de un archivo.

En este ejemplo, la función iniciar() crea una referencia al elemento cajadatos y añade al botón un detector del evento click. Cuando se pulsa el botón Adelante, se ejecuta la función leer() y podemos ver los métodos estudiados previamente en acción.

En primer lugar se declara la dirección URL del archivo a leer. Debido a que aún no hemos estudiado cómo hacer peticiones de origen transversal, este fichero tiene que estar en el mismo dominio (y en este ejemplo, también en el mismo directorio) que el código Javascript.

En el siguiente paso se crea el objeto con el constructor XMLHttpRequest() y es asignado a la variable solicitud. Esta variable se utiliza para añadir un detector al evento cargar, mientras los métodos open() y send() son utilizados para configurar y ejecutar la petición.

Dado que en la solicitud no se enviarán datos, el método send() está vacío (null), pero el método open() necesita sus atributos para configurar la solicitud. Usando este método, declaramos la solicitud como GET, y declaramos también la dirección URL del archivo que será leído y el tipo de operación (true si es asíncrona).

En una operación asíncrona, el navegador seguirá procesando el código mientras

el archivo sea leído. El final de la operación es notificado a través del evento load. Cuando el archivo se ha cargado, este evento se dispara y se llama a la función mostrar().

La función mostrar() comprueba el estado de la operación mediante el valor de la propiedad status y luego inserta el valor de la propiedad responseText en cajadatos para mostrar el contenido del archivo en la pantalla.

Hágalo usted mismo

Para probar este ejemplo, cree un archivo de texto llamado textfile.txt y añada algún contenido a la misma. Descargue este archivo y el resto de los archivos creados con los códigos 21-1, 21-2 y 21-3 a su servidor y abra el documento HTML en su navegador. Después de hacer clic en el botón ¡Adelante!, el contenido del archivo de texto se mostrará en la pantalla.

Conceptos básicos

Los servidores devuelven información para informar del estado de la solicitud. Esta información se denomina código de estado HTTP, e incluye un número y un mensaje que describe el estado. Javascript proporciona dos propiedades para leer esta información: status y statusText La propiedad status devuelve el número que identifica al estado y la propiedad statusText devuelve el mensaje. El valor 200, utilizado en el ejemplo del Código 21-3, es solo uno de varios disponibles e indica que la solicitud ha sido exitosa. El mensaje de este estado es OK. Para obtener una lista completa de los códigos de estado HTTP, visite nuestro sitio web y siga los enlaces de este capítulo.

21.1.2 Propiedades de respuesta

Existen tres tipos de propiedades diferentes que pueden ser utilizadas para obtener información devuelta por una solicitud:

response: Ésta es una propiedad de uso general. Devuelve la respuesta de la solicitud en el formato especificado por el valor de la propiedad responseType. Esta propiedad puede tomar cinco valores diferentes: text, arraybuffer, blob, document o json.

responseTex: Devuelve la respuesta de la solicitud como texto.

responseXML: Devuelve la respuesta de la solicitud como datos XML.

La más útil de estas propiedades para las aplicaciones HTML5 es probablemente response. Esta propiedad devuelve los datos en el formato previamente declarado por la propiedad responseType. La propiedad responseType es parte del objeto XMLHttpRequest y debe ser declarada antes de la solicitud. He aquí un ejemplo práctico:

```
var cajadatos;
function iniciar(){
 cajadatos = document.getElementById('cajadatos');
 var boton = document.getElementById('boton');
 boton.addEventListener('click', leer);
}
function leer(){
 var url = "imagen.jpg";
 var solicitud = new XMLHttpRequest();
 solicitud.responseType = 'blob';
 solicitud.addEventListener('load', mostrar);
 solicitud.open("GET", url, true);
 solicitud.send(null);
}
function mostrar(e){
 var data = e.target;
 if(data.status == 200){
  var imagen = URL.createObjectURL(data.response);
  cajadatos.innerHTML = '<img src="' + imagen + '">';
 }
}
addEventListener('load', iniciar);
```

Código 21-4

Lectura de una imagen desde el servidor utilizando valor blob para la propiedad responseType.

En el Código 21-4, a la propiedad responseType de la solicitud se le asigna el tipo blob. Ahora, los datos recibidos serán un blob que podremos utilizar para almacenar en un archivo, para cortar, cargar, etc. En este ejemplo lo usamos como fuente para un elemento <img> y mostramos así la imagen descargada en la pantalla. Para activar el blob en una URL para la fuente de la imagen, se aplica el método createObjectURL() presentado en el Capítulo 9.

21.1.3 Eventos

Además del evento load, la especificación incluye eventos específicos para el objeto XMLHttpRequest:

loadstart: Este evento se desencadena cuando comienza la solicitud.

progress: Este evento se activa periódicamente durante el envío o la carga de datos.

abort: Este evento se activa cuando la solicitud se cancela.

error: Este evento se activa cuando se produce un error durante la solicitud.

load: Este evento se activa cuando la solicitud se ha completado.

timeout: Si se ha especificado un valor como tiempo de espera, este evento se activa cuando la solicitud no puede ser completada en ese período de tiempo especificado.

loadend: Este evento se activa cuando la solicitud se ha completado (ya sea en caso de éxito o de fracaso).

El acontecimiento más atractivo de todos es, seguramente, progress. Este evento se dispara aproximadamente cada 50 milisegundos para informar al usuario sobre el estado de la petición. Al utilizar el evento progress, se puede notificar al usuario acerca de cada paso del proceso así como crear aplicaciones profesionales de comunicación.

```
var cajadatos;
function iniciar(){
 cajadatos = document.getElementById('cajadatos');
 var boton = document.getElementById('boton');
 boton.addEventListener('click', leer);
}
function leer(){
 var url = "trailer.ogg";
 var solicitud = new XMLHttpRequest();
 solicitud.addEventListener('loadstart', inicio);
 solicitud.addEventListener('progress', estado);
 solicitud.addEventListener('load', mostrar);
 solicitud.open("GET", url, true);
 solicitud.send(null);
}
function inicio(){
 cajadatos.innerHTML = '<progress value="0" max="100">0%</progress>';
}
function estado(e){
 if(e.lengthComputable){
 var per = parseInt(e.loaded / e.total * 100);
 var barraprogreso = cajadatos.querySelector("progress");
 barraprogreso.value = per;
 barraprogreso.innerHTML = per + '%';
 }
}
function mostrar(e){
 var data = e.target;
 if(data.status == 200){
 cajadatos.innerHTML = 'Hecho';
 }
}
addEventListener('load', iniciar);
```

Código 21-5

Mostrar el progreso de la solicitud.

En el Código 21-5 se utilizan tres eventos, loadstart, progress y load para controlar la petición. El evento loadstart llama a la función inicio() para mostrar la barra de progreso en la pantalla por primera vez. Mientras que el archivo es descargado, el evento progress ejecuta la función estado() repetidas veces. Esta función informa al usuario sobre el progreso a través del elemento <progress> y los valores de las propiedades del evento progress, estudiadas en el Capítulo 16.

Importante

Se ha utilizado innerHTML para agregar un nuevo elemento <progress> al documento. Si bien es útil y conveniente para nuestro ejemplo, ésta no es una práctica recomendada. Por lo general, los elementos se agregan al DOM (Document Object Model) utilizando el método de Javascript createElement() junto con appendChild(), como se explica en el Capítulo 4.

Por último, cuando el archivo se ha descargado por completo, el evento load se activa y la función mostrar() muestra el texto Hecho en pantalla.

Hágalo usted mismo

Dependiendo de su conexión a Internet, para ver la barra de progreso, tendrá que utilizar archivos de gran tamaño. En el Código 21-5 declaramos como URL al vídeo trailer.ogg. Usted puede utilizar cualquier otro archivo o descargar el mencionado de www.minkbooks.com/content/.

21.1.4 Envío de datos

Hasta ahora hemos estado recibiendo información desde el servidor, pero no hemos enviado ningún dato o utilizado otro método HTTP que no sea GET. En el siguiente ejemplo vamos a trabajar con el método POST y con un nuevo objeto que nos permitirá enviar información a través de elementos de formularios virtuales.

No hemos mencionado cómo enviar datos a través de un método GET porque es tan simple como añadir los valores de la URL. Tendrá que crear una ruta para la variable url, tal como archivotexto.txt?val1=1&val2=2,y los valores se enviarán junto con la solicitud. Los atributos val1 y val2 de este ejemplo se entenderán como variables GET en el servidor. Por supuesto, un archivo de texto no puede procesar esa información, así que por lo general tendrá un archivo PHP o cualquier otro código del lado del servidor para recibir estos valores. Sin embargo, para las solicitudes POST, no es tan simple.

La solicitud POST incluye no solo toda la información enviada por un método GET, sino también un cuerpo de mensaje que permite enviar cualquier tipo de información, de cualquier longitud. Un formulario HTML suele ser la mejor manera de proporcionar esta información, pero para aplicaciones dinámicas probablemente no sea la mejor opción o la más apropiada. Para resolver este problema, la API incluye la interfaz FormData. Esta sencilla interfaz tiene un constructor y un método para obtener y trabajar con objetos FormData.

FormData(formulario): Este constructor devuelve un objeto FormData utilizado más tarde por el método send() para enviar la información. El atributo formulario es opcional y es una referencia a un formulario HTML. Proporciona una manera fácil de incluir todo un formulario HTML en el

objeto.

append(nombre, valor): Este método agrega datos al objeto FormData. Usa una pareja de palabras clave y valor como atributos. El atributo valor puede ser una cadena o un blob y los datos devueltos representan un campo de formulario.

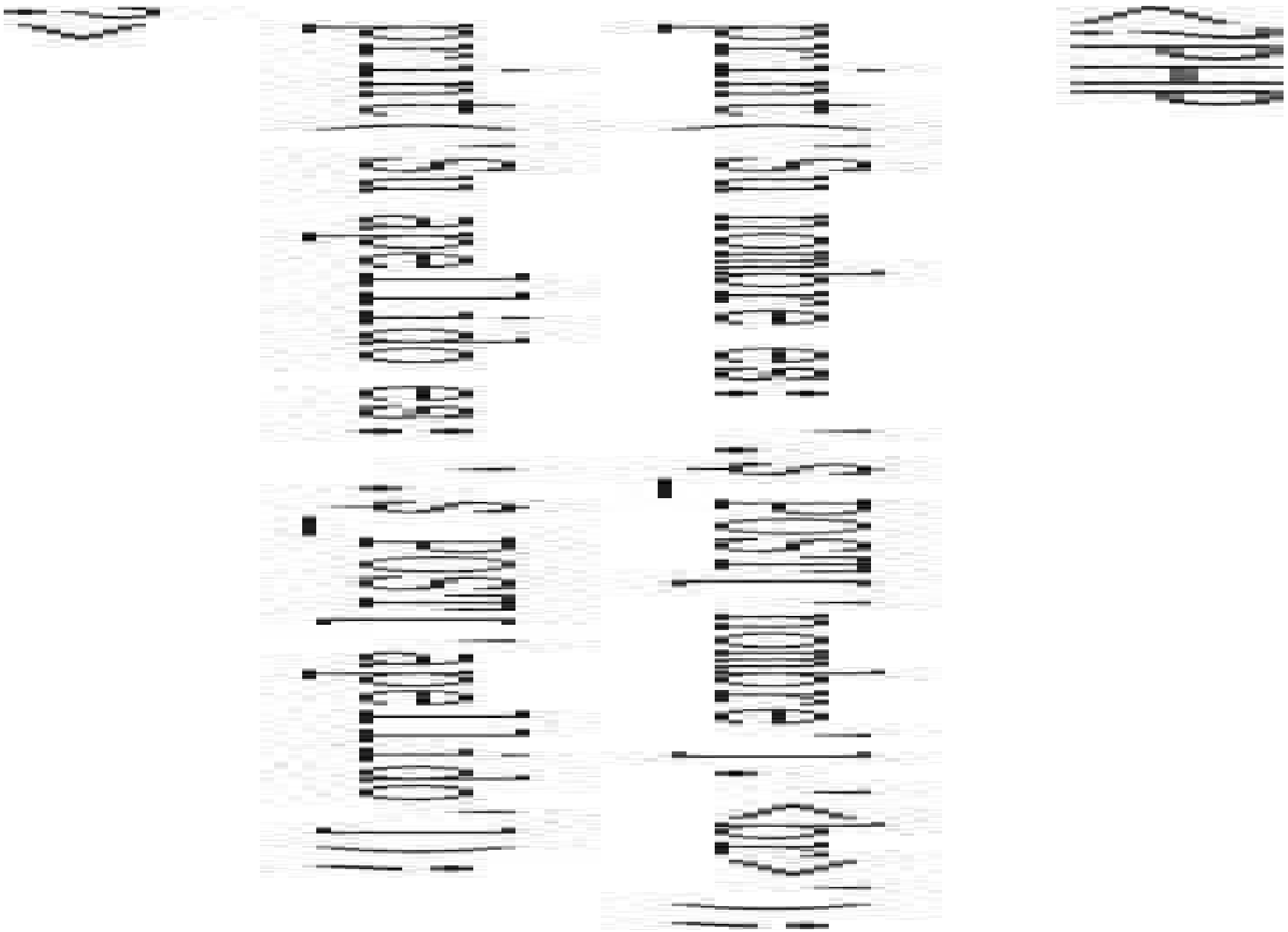
```
var cajadatos;
function iniciar(){
 cajadatos = document.getElementById('cajadatos');
 var boton = document.getElementById('boton');
 boton.addEventListener('click', enviar);
}
function enviar(){
 var data = new FormData();
 data.append('nombre', 'Juan');
 data.append('apellido', 'Diaz');
 var url = "proceso.php";
 var solicitud = new XMLHttpRequest();
 solicitud.addEventListener('load', mostrar);
 solicitud.open("POST", url, true);
 solicitud.send(data);
}
function mostrar(e){
var data = e.target;
if(data.status == 200){
cajadatos.innerHTML = data.responseText;
}
}
addEventListener('load', iniciar);
```

Código 21-6

Envío de un formulario virtual al servidor.

Cuando la información es enviada al servidor, se hace con el objetivo de procesarla y producir un resultado. Por lo general, este resultado se guarda en el servidor y se devuelve alguna información para proporcionar retroalimentación. En el ejemplo de Código 21-6 enviamos datos al archivo process.php y mostramos en pantalla la información devuelta por el código.

Para probar este ejemplo puede imprimir los valores recibidos por process.php con un código como el siguiente:



Código 21-7

Respuesta a una solicitud PUBLICAR (process.php).

Vamos a ver primero cómo se prepara la información para el envío. En la función enviar() del Código 21-6, es invocado el constructor FormData() y el objeto DataForm devuelto se almacena en la variable data. La palabra clave y el valor son añadidos a este objeto con los nombres nombre y apellido mediante el método append(). Estos valores representan los campos de entrada del formulario.

La inicialización de la solicitud es exactamente la misma que en los códigos anteriores excepto que esta vez el primer atributo para el método open() es POST en lugar de GET, y el atributo del método send() es el objeto data.

Cuando el botón Adelante es pulsado, es llamada la función send() y el formulario creado con el objeto FormData se envía al servidor. El archivo process.php recibe estos datos (nombre y apellido) y devuelve al navegador un texto con esta información. La función mostrar() se ejecuta cuando el proceso se ha completado y la información recibida es mostrada en pantalla a través de la propiedad responseText.

Hágalo usted mismo

Para probar este ejemplo necesitará cargar varios archivos a su servidor.

Utilizaremos el mismo documento HTML y los estilos CSS de los códigos 21-1 y 21-2. Las declaraciones Javascript del Código 21-6 sustituyen a las anteriores. También hay que crear un nuevo archivo llamado proceso.php con el Código 21-7 para responder a la solicitud. Suba todos estos archivos a un servidor y abra el documento HTML en su navegador. Al hacer clic en el botón Adelante, debería ser capaz de ver el texto devuelto por el archivo proceso.php en pantalla.

21.1.5 Solicitudes de orígenes cruzados

Hasta ahora hemos trabajado solo con códigos y archivos de datos almacenados en el mismo directorio y el mismo dominio, pero la tecnología CORS estudiada en el Capítulo 10 también se aplica a XMLHttpRequest Level 2, lo que nos permitirá configurar nuestro servidor para que acepte peticiones de origen cruzado.

Importante

El Código 21-8 es un código PHP que solo añade el valor al encabezado devuelto por el archivo process.php. Para incluir esta cabecera en cada archivo enviado por el servidor tendrá que modificar los archivos de configuración de su servidor. Para obtener más información, consulte el Capítulo 10.

Como se estudió en el Capítulo 10, el acceso desde un origen distinto debe ser autorizado en el servidor. La ventaja de esta API es que, en los casos en los que se accede a códigos del lado del servidor, podemos agregar el encabezado al archivo de códigos en lugar de cambiar la configuración del servidor.

La solución para el ejemplo anterior es tan simple como añadir el encabezado Access-Control-Allow-Origin como la primera línea del código PHP.

```php
<?php

header('Access-Control-Allow-Origin: *');

print('Su nombre es: '.$_POST['nombre'].'<br>');

print('Su apellido es: '.$_POST['apellido']);

?>
```

Código 21-8

Permitir múltiples peticiones de origen.

El valor * para el encabezado Access-Control-Allow-Origin representa múltiples orígenes. El Código PHP 21-8 puede ser accesado desde cualquier origen salvo si el valor * es sustituido por un origen específico.

21.5.6 Cargar archivos

La subida de archivos al servidor es una de las principales preocupaciones de los desarrolladores web de hoy. Es una característica requerida por casi todas las aplicaciones en línea que no ha sido considerada por los navegadores hasta ahora. Esta API se encarga de la situación mediante la incorporación de una nueva propiedad que devuelve un objeto XMLHttpRequestUpload no solo para acceder a todos los métodos, propiedades y eventos del objeto XMLHttpRequest sino también para controlar los procesos de carga de datos.

upload: Esta propiedad devuelve un objeto XMLHttpRequestUpload. La propiedad debe ser llamada desde un objeto XMLHttpRequest existente.

Para trabajar con esta propiedad, vamos a necesitar un nuevo documento HTML con un campo <input> que permita seleccionar el archivo que se desea cargar:

```
<!DOCTYPE html>
<html lang="es">
<head>
 <title>Ajax Level 2</title>
 <link rel="stylesheet" href="ajax.css">
 <script src="ajax2.js"></script>
</head>
<body>
 <section id="cajaform">
  <form name="form">
   <label for="archivos">Subir archivo: </label><br>
   <input type="file" name="archivos" id="archivos">
  </form>
 </section>
 <section id="cajadatos"></section>
</body>
</html>
```

Código 21-9

Documento HTML para cargar archivos.

Una forma de cargar un archivo es usar una referencia de archivo y enviarla como un campo de formulario. El objeto FormData estudiado en el Código 21-6 es capaz de manejar esta clase de datos. El sistema detecta automáticamente el tipo de información añadida al objeto FormData y crea las cabeceras apropiadas para la solicitud. El resto del proceso es exactamente el mismo que estudiamos antes en este mismo capítulo.

```
var cajadatos;
function iniciar(){
 cajadatos = document.getElementById('cajadatos');

 var archivos = document.getElementById('archivos');
 archivos.addEventListener('change', cargar);
}
function cargar(e){
 var files = e.target.files;
 var miarchivo = files[0];
 var data = new FormData();
 data.append('file', miarchivo);
 var url = "proceso.php";
 var solicitud = new XMLHttpRequest();
 solicitud.addEventListener('loadstart', inicio);
 solicitud.addEventListener('load', mostrar);
 var xmlupload = solicitud.upload;
 xmlupload.addEventListener('progress', estado);
 solicitud.open("POST", url, true);
 solicitud.send(data);
}
function inicio(){
 cajadatos.innerHTML = '<progress value="0" max="100">0%</progress>';
}
function estado(e){
 if(e.lengthComputable){
  var per = parseInt(e.loaded / e.total * 100);
  var barraprogreso = cajadatos.querySelector("progress");
  barraprogreso.value = per;
  barraprogreso.innerHTML = per + '%';
 }
}
function mostrar(e){
 var data = e.target;
 if(data.status == 200){
  cajadatos.innerHTML = 'Hecho';
 }
}
addEventListener('load', iniciar);
```

Código 21-10

Carga de un archivo con Datos_formulario().

La función principal del Código 21-10 es upload(). La función es llamada cuando el usuario selecciona un archivo nuevo desde el elemento <input> del documento (es decir, cuando el evento change es activado). El archivo seleccionado es recibido y almacenado en la variable miarchivo, exactamente como en la API File en el Capítulo 16, y también en la API Drag and Drop en el Capítulo 13.

Una vez que tenemos una referencia al archivo, es creado el objeto DataForm y el archivo se añade al objeto mediante el método append(). Para enviar el formulario, comenzamos con una solicitud POST y establecemos todos los detectores de eventos para la solicitud, excepto para progress. El evento progress verifica el proceso de carga, así que tenemos que utilizar primero la propiedad upload para obtener el objeto XMLHttpRequestUpload necesario. Después de que este objeto es creado, añadimos el detector para el evento progress y enviamos la solicitud.

El resto del código Javascript lleva a cabo el mismo procedimiento que el ejemplo del Código 21-5, en otras palabras, se muestra en la pantalla una barra de progreso cuando el proceso se inicia y, a continuación, se actualiza de acuerdo con el progreso de este proceso.

21.1.7 Una aplicación real

Cargar un archivo a la vez no es probablemente lo que la mayoría de los desarrolladores web tienen en mente; tampoco desean usar un elemento <input> para seleccionar los archivos que hace falta cargar. Por lo general, todos los programadores quieren que sus aplicaciones sean lo más intuitivas posible, y la mejor manera de hacerlo es mediante la combinación de técnicas y métodos con los que ya están familiarizados. Si sacamos provecho de la API Drag and Drop, crearemos una aplicación de la vida real capaz de cargar varios archivos al servidor al mismo tiempo solo con dejarlos caer en un área de la pantalla.

Para comenzar, vamos a crear un documento HTML con una caja para soltar los archivos:

```
<!DOCTYPE html>
<html lang="es">
<head>
 <title>Ajax Level 2</title>
 <link rel="stylesheet" href="ajax.css">
 <script src="ajax.js"></script>
</head>
<body>
 <section id="cajadatos">
  <p>Arrastre sus archivos aquí</p>
 </section>
</body>
</html>
```

Código 21-11

Área para soltar los archivos a cargar.

El código Javascript de este ejemplo combina dos API y trabaja constantemente con funciones anónimas para mantener todo organizado y dentro del mismo ámbito (dentro de la misma función). Hay que tomar los archivos arrastrados al elemento cajadatos y hacer una lista en pantalla, preparar el formulario con el archivo que se desea enviar, hacer una petición de carga para cada archivo y, finalmente, actualizar la barra de progreso mientras los archivos se están cargando.

```
var cajadatos;
function iniciar(){
 cajadatos = document.getElementById('cajadatos');
 cajadatos.addEventListener('dragenter', function(e){
                        e.preventDefault();
});
 cajadatos.addEventListener('dragover', function(e){
                        e.preventDefault();
});
 cajadatos.addEventListener('drop', soltar);
}
function soltar(e){
 e.preventDefault();
 var archivos = e.dataTransfer.archivos;
 if(archivos.length){
   var lista = '';
   for(var f = 0; f < archivos.length; f++){
     var archivo = archivos[f];
     lista += '<div>archivo: ' + archivo.name;
     lista += '<br><span><progress value="0" max="100">0%</progress>
                        </span>';
     lista += '</div>';
   }
   cajadatos.innerHTML = lista;
   var contar = 0;
   var cargar = function(){
     var miarchivo = archivos[contar];
     var data = new FormData();
     data.append('archivo', miarchivo);
     var url = "process.php";
     var solicitud = new XMLHttpRequest();
     var xmlcargar = solicitud.cargar;

    xmlcargar.addEventListener('progress', function(e){
     if(e.lengthComputable){
       var child = contar + 1;
       var per = parseInt(e.loaded / e.total * 100);
       var barraprogreso = cajadatos.querySelector("div:nth-child
                        ("+child+") > span > progress");
       barraprogreso.value = per;
       barraprogreso.innerHTML = per + '%';
     }
   });
   solicitud.addEventListener('load', function(){
    var child = contar + 1;
    var elem = databox.querySelector("div:nth-child(" + child + ") >
                              span");
    elem.innerHTML = 'Hecho';

    contar++;

    if(contar < archivos.length){
      cargar();
    }
   });
   solicitud.open("POST", url, true);
   solicitud.send(data);
  }
  cargar();
 }
}
addEventListener('load', iniciar);
```

Código 21-12

Cargar archivos uno a uno.

Bueno, el Código 21-12 no es fácil de seguir, pero será más fácil si lo hacemos paso a paso. Como de costumbre, todo comienza a partir de la función iniciar(), que es llamada cuando se ha cargado el documento. Esta función crea una referencia al elemento cajadatos, que será nuestro buzón en este ejemplo, y añade detectores para tres eventos que controlarán la operación de arrastrar y soltar. Para saber más acerca de la API Drag and Drop, consulte el Capítulo 13. Básicamente, el evento dragenter se activa cuando los archivos que están siendo arrastrados entran en el área del buzón, el evento dragover se activa periódicamente mientras los archivos se encuentran sobre el área del buzón y, finalmente, el evento drop se activa cuando los archivos se dejan caer en el buzón. No tenemos que hacer nada con dragenter y dragover en este ejemplo, por lo que estos eventos son cancelados para prevenir el comportamiento por defecto del navegador. El único evento al que tenemos que responder es drop. La función soltar() ajusta el detector para este evento y se ejecuta cada vez que algo cae en el elemento cajadatos.

La primera línea en la función soltar() también utiliza el método preventDefault() pues lo que queremos hacer con los archivos arrastrados no es lo que el navegador haría de forma predeterminada. Ahora que tenemos el control absoluto de la situación, es el momento de procesar los archivos soltados. En primer lugar, obtenemos los archivos desde el objeto dataTransfer. El valor devuelto es una matriz de archivos que se almacenan en la variable archivos. Para asegurarnos de que los archivos (y no otro tipo de elementos) han sido arrastrados a nuestro buzón, comprobamos el valor de la propiedad length con la declaración if(files.length). Si este valor es distinto a 0 o null, significa que ha sido arrastrado al menos un archivo y, por tanto, podemos seguir adelante.

Ahora es el momento de trabajar con los archivos recibidos. Con un bucle for, navegamos a través de la matriz archivos y creamos una lista de elementos que contienen el nombre del archivo y una barra de progreso encerrada entre etiquetas . Una vez que la lista ha sido acabada, el resultado se muestra en la pantalla dentro de cajadatos.

Parece que la función soltar() hace todo el trabajo, pero dentro de esta función hay otra llamada cargar() que maneja el proceso de carga de cada archivo. Así que después de mostrar los archivos en la pantalla, el siguiente paso es crear esta función y llamarla para cada archivo en la lista.

La función cargar() se crea utilizando una función anónima. Dentro de esta función, primero se selecciona un archivo de la matriz usando la variable contar como un índice. Esta variable tiene inicialmente un valor igual a 0, por lo que la primera vez que se llama a la función cargar(), el primer archivo de la lista es seleccionado y cargado.

Para cargar cada archivo, se utiliza el mismo método que en los ejemplos anteriores. Se almacena una referencia al archivo en la variable miarchivo; se crea un objeto FormData usando el constructor FormData(), y el archivo se agrega al objeto usando el método append().

Esta vez, solo detectamos dos eventos para controlar el proceso: progress y load. Cada vez que el evento progress se activa, se llama una función anónima que actualiza el estado de la barra de progreso para el archivo que se está cargando. Para identificar el elemento <progress> que corresponde a dicho archivo, se utiliza el método querySelector() con la pseudo-clase :nth-child(). El índice de la pseudo-clase es calculado utilizando el valor de la variable contar. Esta variable tiene el número de índice de la matriz archivos, pero mientras este índice comienza en 0, el índice utilizado para la lista de los elementos anidados al que se accede con :nth-child() comienza en el valor 1. Para obtener el valor del índice correspondiente al elemento <progress> adecuado, sumamos 1 al valor la

variable contar, almacenamos el resultado en la variable child y utilizamos esta variable como índice.

Cada vez que finaliza el proceso anterior, tenemos que informar de la situación y pasar al siguiente archivo de la matriz archivos. Para este propósito, en la función anónima ejecutada al dispararse el evento load, sumamos 1 al valor de contar, sustituimos el elemento <progress> por la cadena Hecho y llamamos a la función cargar() de nuevo en caso de que haya más archivos que deban ser procesados.

Si usted sigue el script línea a línea, verá que la función upload(), después de haber sido declarada, no vuelve a ser mencionada hasta el final de la función soltar(). Debido a que el valor de contar ha sido establecido previamente como 0, el primer archivo de la matriz archivos se procesará primero. Entonces, cuando el proceso de carga de este archivo se ha completado, el evento load se activará y la función anónima llamada para manejar este evento aumentará en 1 el valor de contar y ejecutará la función cargar() de nuevo, para procesar el siguiente archivo de la matriz. Al final, todos los archivos soltados en el área del buzón habrán sido subidos al servidor, uno a uno.

22 Web Messagin API

22.1 Mensajería entre documentos

La API Web Messaging permite que aplicaciones de orígenes diferentes se comuniquen entre sí. Llamaremos a este proceso “mensajería entre documentos”. Las aplicaciones ejecutadas en diferentes marcos, pestañas o ventanas ahora pueden comunicarse gracias a esta tecnología. El procedimiento es simple: enviamos un mensaje desde un archivo y este mensaje es procesado en el documento de destino.

22.1.1 Enviar un mensaje

Para enviar mensajes, la API proporciona el método postMessage():

postMessage(mensaje, destino): Este método envía un mensaje a otro documento. El atributo mensaje es una cadena de texto que representa el mensaje que debe ser transmitido, y el atributo detino es el dominio del documento de destino (nombre del host o del puerto, como veremos más adelante). El objetivo puede ser declarado como un dominio específico, como cualquier otro documento con la tecla estrella (*), o como el mismo que el origen usando la barra inclinada (/). El método también puede incluir un conjunto de puertos como un tercer atributo.

El método de comunicación es asíncrono. Para detectar los mensajes enviados por un documento en particular, la API utiliza el evento message. que incluye una serie de propiedades para devolver la información:

data: Esta propiedad del evento message devuelve el contenido del mensaje.

origin: Esta propiedad del evento message devuelve el origen del documento que envió el mensaje. Este valor, que por lo general lleva el nombre de host, puede ser utilizado para enviar un mensaje de vuelta.

source: Esta propiedad del evento message devuelve un objeto para identificar la fuente del mensaje. Este valor puede ser utilizado como una referencia al remitente para responder el mensaje, como se verá más

adelante.

22.1.2 Comunicar con un iframe

Para trabajar con un ejemplo de esta API debemos tener en cuenta que el proceso de comunicación se produce entre diferentes ventanas, marcos y pestañas. Por tanto, debe proporcionar documentos para cada lado involucrado. Nuestro ejemplo incluye un documento HTML con un marco incrustado (un elemento iframe), los códigos Javascript adecuados para el documento principal y el documento que se colocará dentro del marco. De momento, vamos a empezar con el documento HTML principal.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mensajería entre documentos</title>
  <link rel="stylesheet" href="messaging.css">
  <script src="messaging.js"></script>
</head>
<body>
  <section id="cajaformulario">
    <form name="form">
      <label for="nombre">Su nombre: </label>
      <input type="text" name="nombre" id="nombre" required>
      <input type="button" id="boton" value="Enviar">
    </form>
  </section>
  <section id="cajadatos">
    <iframe id="iframe" src="iframe.html" width="500" height="350">
                        </iframe>
  </section>
</body>
</html>
```

Código 22-1

Documento principal con un iframe.

Como usted puede ver, hay dos elementos <section> como en los documentos anteriores, pero esta vez cajadatos incluye un elemento <iframe> en el que se carga el archivo iframe.html. Volveremos a este punto más tarde.

Conceptos básicos

El elemento <iframe> sustituye al obsoleto sistema de etiquetas <frameset> y <frame>. Este elemento nos permite insertar un documento en otro archivo. El documento que se incrustará en el <iframe> se declara con el atributo src. Todo lo que esté dentro del iframe responde tal como si se encontrara en la ventana principal.

Vamos a añadir algunos estilos a la estructura:

```
 #cajaformulario{
  float: left;
  padding: 20px;
  border: 1px solid #999999;
}

#cajadatos{
  float: left;
  width: 500px;
  margin-left: 20px;
   padding: 20px;
  border: 1px solid #999999;
}
```

Código 22-2

Aplicar estilo a las cajas (messaging.css).

El código Javascript del documento principal toma el valor de la entrada nombre del formulario y lo envía al documento que se encuentra dentro del iframe usando el método postMessage().

```
function iniciar(){
  var boton = document.getElementById('boton');
  boton.addEventListener('click', enviar);
}
function enviar(){
  var nombre = document.getElementById('nombre').value;
  var iframe = document.getElementById('iframe');
  iframe.contentWindow.postMessage(nombre, '*');
}
addEventListener('load', iniciar);
```

Código 22-3

Enviar un mensaje al iframe (messaging.js).

En el Código 22-3, el mensaje está formado por el valor de la entrada nombre. El símbolo * se utiliza como objetivo para enviar el mensaje a todos los documentos que se ejecuten dentro del iframe, independientemente de su origen.

Conceptos básicos

El método postMessage() pertenece al objeto Window. Para obtener el objeto Window de un iframe del documento principal, tenemos que utilizar la propiedad contentWindow.

Una vez que se hace clic en el botón Enviar del documento principal, es llamada la función send()y se envía el valor del campo de entrada al interior del iframe. Ahora es el momento de tomar este mensaje del iframe y procesarlo. Vamos a crear un pequeño documento HTML para el iframe, para que esta información sea mostrada en la pantalla.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Ventana del iframe</title>
  <script src="iframe.js"></script>
</head>
<body>
  <section>
    <div><b>Mensaje de la ventana principal:</b></div>
    <div id="cajadatos"></div>
  </section>
</body>
</html>
```

Código 22-4

Documento HTML para el iframe (iframe.html).

Este documento tiene su propio elemento cajadatos, que utilizaremos para mostrar el mensaje en la pantalla, y su propio código Javascript para procesar el mensaje.

```
function iniciar(){
  addEventListener('message', receptor);
}
function receptor(e){
  var cajadatos = document.getElementById('cajadatos');
  cajadatos.innerHTML = 'Mensaje de: ' + e.origin + '<br>';
  cajadatos.innerHTML += 'Mensaje: ' + e.data;
}
addEventListener('load', iniciar);
```

Código 22-5

Procesamiento de mensajes en el objetivo (iframe.js)

Como hemos explicado antes, para detectar los mensajes, la API proporciona el evento message y algunas propiedades. En el Código 22-5, la función receptor() es llamada cuando se activa el evento message. Esta función muestra el contenido del mensaje utilizando la propiedad datos y la información sobre el documento que envió el mensaje, esto último gracias al valor de origen.

Recuerde que este código Javascript pertenece al documento HTML del iframe y no al documento principal del Código 22-1. Se trata de dos documentos diferentes con su propio entorno, alcance y scripts: uno está en la ventana principal del navegador y el otro está dentro del iframe.

Hágalo usted mismo

Deberá crear y cargar a un servidor un total de cinco archivos con el fin de probar el ejemplo anterior. En primer lugar, cree un nuevo archivo HTML con el Código 22-1 para el documento principal. Este documento requiere el archivo messaging.css con los estilos del Código 22-2 y el archivo messaging.js con las declaraciones Javascript del Código 22-3. El documento del Código 22-1 tiene un <iframe> con el elemento iframe.html como fuente. Tendrá que crear este archivo con las declaraciones HTML del Código 22-4 y el correspondiente archivo Javascript iframe.js con el Código 22-5. Suba todos los archivos a su servidor, abra el primer documento HTML en el navegador y envíe su nombre al iframe utilizando el formulario.

22.1.3 Filtros y orígenes cruzados

Hasta ahora nuestro método de codificación no ha seguido lo que se considera la mejor práctica, sobre todo en lo que respecta a cuestiones de seguridad. El código Javascript del documento principal envía el mensaje al iframe indicado pero no se controla el tipo de documento permitido. De hecho, cualquier documento dentro del iframe será capaz de leer el mensaje. Además, el código del iframe no controla el origen y procesa cualquier mensaje recibido. Ambas partes del proceso de comunicación deben mejorar para evitar abusos.

En el siguiente ejemplo, vamos a corregir esta situación y le mostrará cómo responder a un mensaje del destino con otra propiedad del evento message llamada source.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mensajería entre documentos</title>
  <link rel="stylesheet" href="messaging.css">
  <script src="messaging.js"></script>
</head>
<body>
  <section id="cajaformulario">
    <form name="form">
      <label for="nombre">Tu nombre: </label>
      <input type="text" name="nombre" id="nombre" required>
      <input type="button" id="boton" value="Enviar">
    </form>
  </section>
  <section id="cajadatos">
    <iframe id="iframe" src="http://dominio2.com/iframe.html"
                                width="500" height="350"></iframe>
  </section>
</body>
</html>
```

Código 22-6

Comunicar orígenes y destinos específicos.

En el último documento no solo proporcionamos el archivo HTML fuente del iframe, como hicimos antes, sino que también declaramos la ruta completa a una ubicación diferente (www.dominio2.com). El documento principal, por ejemplo, estará en www.dominio1.com y el contenido del iframe estará en www.dominio2.com. Para evitar abusos, vamos a tener que declarar estos lugares e indicar quién puede leer un mensaje y desde dónde puede hacerlo. El siguiente código Javascript considera esta situación:

```
function iniciar(){
  var boton = document.getElementById('boton');
  boton.addEventListener('click', enviar);
  addEventListener('message', receptor);
}
function enviar(){
  var nombre = document.getElementById('nombre').value;
  var iframe = document.getElementById('iframe');
  iframe.contentWindow.postMessage(nombre, 'http://dominio2.com');
}
function receptor(e){
  if(e.origin == 'http://dominio2.com'){
    document.getElementById('nombre').value = e.data;
  }
}
addEventListener('load', iniciar);
```

Código 22-7

Comunicación con orígenes específicos (messaging.js).

Preste atención a la función enviar() del Código 22-7, donde el método postMessage()ahora declara el objetivo específico para el mensaje (http://dominio2.com) Solo los documentos que estén dentro del iframe y desde ese origen específico podrán leer este mensaje.

En la función iniciar() del Código 22-7 añadimos también un detector para el evento message. El propósito de la función receptor() es detectar la respuesta enviada por el documento del iframe. Más adelante comprenderá para qué lo hemos hecho de esta manera.

Ahora vamos a ver el código Javascript del iframe, para ver cómo se procesa un mensaje de un origen específico y cómo se responde a este mensaje. Utilizaremos exactamente el mismo documento HTML del Código 22-4 para el iframe.

```
function iniciar(){
  addEventListener('message', receptor);
}
function receptor(e){
  var cajadatos = document.getElementById('cajadatos');
  if(e.origin == 'http://www.dominio1.com'){
    cajadatos.innerHTML = 'valid message: ' + e.data;
    e.source.postMessage('mensaje recibido', e.origin);
  }else{
    cajadatos.innerHTML = 'origen inválido';
  }
}
addEventListener('load', iniciar);
```

Código 22-8

Responder al documento principal (iframe.js).

El funcionamiento del filtro para el origen es tan simple como comparar el valor de la propiedad origin con el dominio del que deseamos leer los mensajes. Una vez que se comprueba que el origen es válido, el mensaje se muestra en la pantalla y, a continuación, se envía una respuesta utilizando el valor de la propiedad source. La propiedad origen también se utiliza para declarar esta respuesta, que solo está disponible para la ventana que ha enviado el mensaje. Ahora puede volver al Código 22-7 y comprobar cómo la función receptor() procesa esta respuesta.

Hágalo usted mismo

Este ejemplo es un poco complicado. Como estamos utilizando dos orígenes diferentes, necesita dos dominios (o subdominios) diferentes para probar los códigos. Reemplace los dominios indicados en los códigos por los suyos, a continuación cargue los códigos para el documento principal en un dominio y los códigos para el iframe en el otro y podrá ver cómo se comunican estos dos documentos de diferentes dominios.

23 API WebSocket

23.1 WebSockets

La API WebSocket proporciona soporte a la conexión para lograr una comunicación bidireccional entre navegadores y servidores más rápida y más eficaz. La conexión se realiza a través de un TCP sin enviar cabeceras HTTP, lo que reduce el tamaño de los datos transmitidos en cada llamada. La conexión es también persistente, lo que permite a los servidores mantener a los clientes actualizados sin una solicitud previa, de manera que no es necesario llamar al servidor por actualizaciones. De hecho, el propio servidor envía automáticamente información sobre su estado actual.

WebSocket podría parecer una versión mejorada de Ajax, pero es una alternativa totalmente diferente de comunicación que permite la construcción de aplicaciones en tiempo real en una plataforma escalable, como pueden ser, los videojuegos de varios jugadores, los chats, etc.

La API es simple; incluye algunos métodos y eventos para abrir y cerrar la conexión y enviar y detectar mensajes. Sin embargo, ningún servidor proporciona este servicio de forma predeterminada y la respuesta tiene que ser adaptada a nuestras necesidades, así que tenemos que instalar nuestro propio servidor WS (servidor WebSocket) para poder establecer la comunicación entre el navegador y el servidor que aloje a nuestra aplicación.

23.1.1 Servidor WebSocket

Si es un experto programador, es probable que pueda encontrar la manera de construir su propio código de servidor WS, pero para aquellos que quieren invertir su tiempo en otras actividades, hay varios códigos disponibles para configurar un servidor WS y poder procesar peticiones WS. Dependiendo de sus preferencias, puede optar por códigos hechos en PHP, Java y Ruby entre otros. Para obtener una lista completa, visite nuestra página web y siga los enlaces de este capítulo.

En este libro vamos a mostrar cómo instalar un servidor PHP. Hay varias versiones programadas en este lenguaje pero vamos a utilizar una sencilla llamada phpws.

Importante

Para el momento de escribir este libro, solo Google Chrome permite las conexiones WebSocket por defecto. Le recomendamos probar estos ejemplos en Google Chrome y también buscar en Internet otras versiones de servidores WS disponibles. El servidor phpws requiere al menos la versión PHP 5.3 para funcionar correctamente y la cuenta de hospedaje debe incluir el acceso shell para que pueda comunicarse con el servidor y ejecutar el código PHP. Puede solicitar a su proveedor de host que permita el acceso shell si no lo tiene por defecto.

23.1.2 Instalación y ejecución de un servidor WS

El código phpws incluye varios archivos para crear las clases y los métodos necesarios para correr el servidor WS. La biblioteca está disponible en http://code.google.com/p/phpws/, pero para probar estos ejemplos, lo único que tiene que hacer es descargar el paquete que preparamos para este capítulo en www.minkbooks.com/content/. Baje el archivo ws.rar, descomprímalo y cargue la carpeta ws en el servidor usando cualquier programa de FTP (por ejemplo, FileZilla). Dentro de esta carpeta encontrará el archivo server.php con una modificación del archivo original llamado demo.php. Hemos añadido unas líneas de código al archivo para adaptar el código a nuestros ejemplos. Puede cambiar el código más adelante para adaptarlo a sus propias necesidades. Aquí está la función que usaremos para procesar los mensajes:

```
public function onMessage(IWebSocketConnection $user,
                IWebSocketMessage $msg){
  $msg = trim($msg->getData());
  switch($msg){
   case 'hola':
    $msgback = WebSocketMessage::create("hola humano");
    $user->sendMessage($msgback);
    break;
   case 'nombre':
    $msgback = WebSocketMessage::create("No tengo nombre");
    $user->sendMessage($msgback);
    break;
   case 'edad':
    $msgback = WebSocketMessage::create("Soy viejo");
    $user->sendMessage($msgback);
    break;
   case 'fecha':
    $msgback = WebSocketMessage::create("Hoy es ".date("F j, Y"));
    $user->sendMessage($msgback);
    break;
   case 'hora':
    $msgback = WebSocketMessage::create("La hora es ".date("H:iA"));
    $user->sendMessage($msgback);
    break;
   case 'gracias':
    $msgback = WebSocketMessage::create("Por nada");
    $user->sendMessage($msgback);
    break;
   case 'adios':
    $msgback = WebSocketMessage::create("Hasta luego");
    $user->sendMessage($msgback);
    break;
   default:
    $msgback = WebSocketMessage::create("I don't understand");
    $user->sendMessage($msgback);
    break;
   }
}
```

Código 23-1

Función de PHP adaptada a nuestros ejemplos (server.php)

Una vez que tenga todos estos archivos en su servidor, es el momento de ejecutar el código. WebSocket utiliza una conexión persistente, por lo que el servidor WS debe estar en funcionamiento todo el tiempo, capturando peticiones y enviando actualizaciones a los usuarios. Para ejecutar el archivo PHP, puede acceder a su servidor mediante una conexión SSH, encontrar la carpeta ws e introducir el comando php server.php.

Conceptos básicos

SSH es un protocolo de red (Secure Shell) que puede ser utilizado para acceder a su servidor y controlarlo de forma remota. Le permitirá trabajar con carpetas y archivos de su servidor y ejecutar códigos. Una de las aplicaciones gratuitas más populares que ofrecen una consola para el acceso Shell se llama PuTTY y está disponible en www.chiark.greenend.org.uk/~sgtatham/putty/). Después de ejecutar PuTTY e insertar el dominio de su servidor aparecerá una consola de texto que le pedirá su nombre de usuario y contraseña. Una vez dentro del servidor, tendrá acceso a la carpeta ws y podrá ejecutar el código tal como se ha explicado anteriormente.

El servidor WS ya está está preparado y en funcionamiento.. Ahora es el momento de conectar con el servidor de nuestra página web.

23.1.3 Constructor

Antes de programar el código Javascript para interactuar con el servidor WS, vamos a ver lo que la API proporciona para este propósito. La especificación declara una interfaz con algunos métodos, propiedades y eventos, además de un constructor para establecer la conexión:

WebSocket(url): Este constructor inicia una conexión entre la aplicación y el servidor WS dirigida por el atributo url. Devuelve un objeto WebSocket que hace referencia a esta conexión. Puede ser especificado un segundo atributo para proporcionar una matriz con subprotocolos de comunicación.

23.1.4 Métodos

La conexión es iniciada por el constructor, por lo que solo hay dos métodos para trabajar con ella:

send(datos): Este método es necesario para enviar un mensaje al servidor WS. El atributo datos representa una cadena de texto con la información que debe ser transmitida.

close(): Este método cierra la conexión.

23.1.5 Propiedades

Algunas propiedades nos informan acerca de la configuración y el estado de la conexión:

url: Esta propiedad devuelve la dirección URL a la que está conectada la aplicación.

protocol: Esta propiedad devuelve el subprotocolo utilizado, si lo hubiere.

readyState: Esta propiedad devuelve un número que representa el estado de la conexión: 0 si la conexión no se ha establecido aún, 1 si el enlace se ha abierto, 2 si la conexión está siendo cerrada y 3 si la conexión ya ha sido cerrada.

bufferedAmount: Ésta es una propiedad muy útil que permite conocer los datos solicitados que aún no se han enviado al servidor. El valor devuelto ayuda a regular la cantidad de datos y la frecuencia de las peticiones con el fin de no saturar el servidor.

23.1.6 Eventos

Para conocer el estado de la conexión y detectar los mensajes enviados por el servidor, tenemos que utilizar eventos. La API proporciona los siguientes:

open: Este evento se activa cuando se abre la conexión.

message: Este evento se activa cuando hay un mensaje del servidor disponible.

error: Este evento se activa cuando se produce un error.

close: Este evento se activa cuando se cierra la conexión.

23.1.7 Documento HTML

El archivo server.php tiene un método llamado onMessage() que procesa una pequeña lista de comandos predefinidos y devuelve la respuesta correcta (vea el Código 23-1). Para propósitos de prueba, vamos a utilizar un formulario para insertar y enviar estos comandos al servidor:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>WebSocket</title>
  <link rel="stylesheet" href="websocket.css">
  <script src="websocket.js"></script>
</head>
<body>
  <section id="cajaformulario">
    <form name="form">
      <label for="command">Command: </label><br>
      <input type="text" name="command" id="command"><br>
      <input type="button" id="button" value="Enviar">
    </form>
  </section>
  <section id="cajadatos"></section>
 </body>
 </html>
```

Código 23-2

Documento HTML para insertar comandos.

También necesitaremos un archivo CSS con los siguientes estilos, llamado websocket.css:

```
#cajaformulario{
   float: left;
   padding: 20px;
   border: 1px solid #999999;
}
#cajadatos{
   float: left;
   width: 500px;
   height: 350px;
   overflow: auto;
   margin-left: 20px;
   padding: 20px;
   border: 1px solid #999999;
}
```

Código 23-3

Estilos para las cajas.

23.1.8 Iniciar la comunicación

Como siempre, el código Javascript es responsable de todo el proceso. Vamos a crear nuestra primera aplicación de comunicación para comprobar cómo funciona la API.

```
var cajadatos, socket;
function iniciar(){
  cajadatos = document.getElementById('cajadatos');
  var button = document.getElementById('button');
  button.addEventListener('click', send);
  socket = new WebSocket("ws://SU_IP:12345/ws/server.php");
  socket.addEventListener('message', received);
}
function received(e){
  var list = cajadatos.innerHTML;
  cajadatos.innerHTML = 'Received: ' + e.data + '<br>' + list;
}
function send(){
  var command = document.getElementById('command').value;
  socket.send(command);
}
addEventListener('load', iniciar);
```

Código 23-4

Envío de mensajes al servidor.

Importante

Reemplace en el código el texto SU_IP por la IP de su servidor. También puede utilizar su dominio, pero el uso de la IP tiene el propósito de evitar la traducción DNS. Deberá usar esta técnica siempre que quiera tener acceso a su aplicación evitando el tiempo empleado por la red para traducir el dominio a la dirección IP correspondiente.

En la función iniciar() del Código 23-4, el objeto WebSocket se construye y almacena en la variable socket. El atributo url señala la ubicación del archivo server.php en el servidor. Además, el puerto de conexión es declarado en esta URL. Por lo general, el host es especificado con el número IP del servidor, y el valor del puerto es 8000 o 8080, pero eso depende de sus necesidades, la configuración de su servidor, los puertos disponibles, la ubicación del archivo en su servidor, etc.

Importante

La función que hemos preparado para estos ejemplos comprueba el mensaje

recibido y compara su valor con una lista de comandos predefinidos. Los comandos disponibles son hola, nombre, edad, fecha, hora, gracias y bye. Por ejemplo, si introduce el comando hola en el formulario, el servidor devolverá el mensaje Hola humano. Puede cambiar esta función para satisfacer sus necesidades. Estudie la secuencia de comandos del Código 23-1 para entender cómo funciona esta función.

Después de obtener el objeto WebSocket, se añade un detector para el evento message al mismo. Cada vez que el servidor WS envía un mensaje al navegador, se dispara el evento message y es invocada la función received() para controlar el evento. Al igual que en las API anteriores, este evento incluye la propiedad datos con el contenido del mensaje. En la función received() se utiliza esta propiedad para mostrar el mensaje en la pantalla.

Hemos incluido la función la función send()para enviar mensajes al servidor. El valor del elemento <input> denominado command es tomado por esta función y enviado al servidor WS utilizando el método send().

23.1.9 Aplicación completa

En el último ejemplo puede ver fácilmente cómo funciona el proceso de comunicación para esta API. La conexión es iniciada por el constructor WebSocket(), el método send() envía cada mensaje que queremos que sea procesado por el servidor, y el evento message informa a la aplicación de la llegada de nuevos mensajes desde el servidor. Sin embargo, no se cierra la conexión, no se comprueba si hay errores ni se detecta si la conexión está lista para trabajar. Veamos ahora un ejemplo que utiliza todos los eventos proporcionados por esta API para informar al usuario sobre el estado de la conexión en cada paso del proceso.

```
var cajadatos, socket;
function iniciar(){
  cajadatos = document.getElementById('cajadatos');
  var button = document.getElementById('button');
  button.addEventListener('click', send);
  socket = new WebSocket("ws://TU_IP:12345/ws/server.php");
  socket.addEventListener('open', opened);
  socket.addEventListener('message', received);
  socket.addEventListener('close', closed);
  socket.addEventListener('error', showerror);
}
function opened(){
  cajadatos.innerHTML = 'CONEXIÓN ABIERTA<br>';
  cajadatos.innerHTML += 'Status: ' + socket.readyState;
}
function received(e){
  var list = cajadatos.innerHTML;
  cajadatos.innerHTML = 'Received: ' + e.data + '<br>' + list;
}
function closed(){
  var list = cajadatos.innerHTML;
  cajadatos.innerHTML = 'CONEXIÓN CERRADA<br>' + list;
  var button = document.getElementById('button');
  button.disabled = true;
}
function showerror(){
  var list = cajadatos.innerHTML;
  cajadatos.innerHTML = 'ERROR<br>' + list;
}
function send(){
  var command = document.getElementById('command').value;
  if(command == 'cerrar'){
    socket.close();
  }else{
    socket.send(command);
  }
}
addEventListener('load', iniciar);
```

Código 23-5

Informar al usuario sobre el estado de la conexión.

Hay algunas mejoras en el Código 23-5 respecto al ejemplo anterior. Hemos añadido detectores de todos los eventos disponibles en el objeto WebSocket y se han creado funciones adecuadas para manejar estos eventos. También se muestra el estado cuando se abre la conexión mediante el valor de la propiedad readyState, se cierra la conexión con el método close() cuando el comando cerrar es enviado desde el formulario y se desactiva el botón Enviar cuando la conexión es cerrada (button.disabled = True).

Hágalo usted mismo

Este último ejemplo requiere el documento HTML y los estilos CSS de los códigos 23-2 y 23-3. Abra el archivo HTML en su navegador, introduzca un comando en el formulario y haga clic en el botón Enviar. Debería obtener una respuesta desde el servidor de acuerdo con el comando introducido (hola, nombre, edad, fecha, hora, gracias o adios). Envíe el comando cerrar para cerrar la conexión.

Importante

Recuerde que su servidor WS debe estar en funcionamiento todo el tiempo para procesar las solicitudes.

24 API WebRTC

24.1 Llega la revolución

Cada API introducida en HTML5 tiene sus propias características: hay algo único en cada una de ellas que puede ser descrito con un solo adjetivo. Algunas son las más bellas, otras las más complejas, algunas las más interesantes. Y la API WebRTC que presentamos en este capítulo se resume en la palabra revolucionario. Si tenemos que elegir una sola API para crear aplicaciones revolucionarias, ésta es WebRTC.

24.1.1 El viejo paradigma

WebRTC significa “Comunicación en tiempo real” (Web Real-Time Communication). Ya no se trata solo de comunicación sino más bien de un nuevo tipo de comunicación específica para la Web. Esta API permite a los desarrolladores crear aplicaciones que conectan a los usuarios entre sí, sin intermediarios. Las aplicaciones que implementan WebRTC pueden transmitir vídeo, audio y datos directamente de un usuario a otro, creando una eficaz red entre pares. Este es un gran cambio de paradigma. Hasta ahora, los usuarios podían compartir información en la Web a través de un servidor. Los servidores eran como grandes repositorios de contenido, accesibles por un dominio o una dirección IP. Los usuarios tenían que conectarse a estos servidores, descargar o cargar información, y esperar a que otros usuarios hicieran lo mismo del otro lado. Si queríamos compartir una imagen con un amigo, por ejemplo, teníamos que subir la imagen a un servidor, y esperar a que nuestro amigo se conectase al mismo servidor y descargara la imagen en su ordenador. El proceso se ilustra en la Figura 24-1:

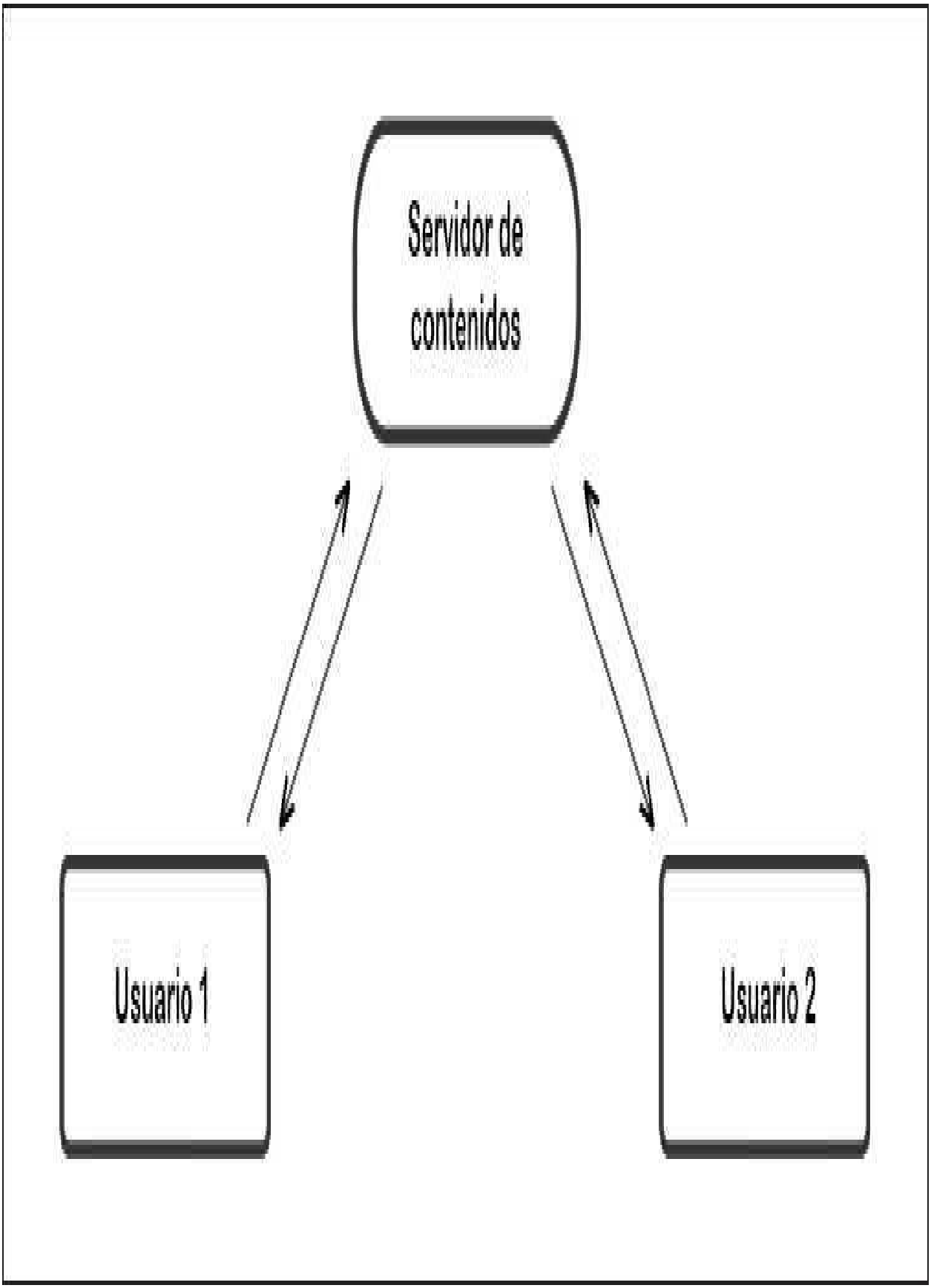

Servidor de contenidos
Usuario 1
Usuario 2

Figura 24-1

El viejo paradigma.

No había manera de que el Usuario 1 enviara información al Usuario 2 sin utilizar un servidor. Cada proceso requería un servidor que almacenara la información enviada por un usuario y la proporcionara (o sirviera) al otro. Había una gran cantidad de aplicaciones fuera de la Web que conectaban directamente a los usuarios entre sí, lo que les permitía enviar mensajes instantáneos o realizar videollamadas, pero el navegador era incapaz de hacerlo.

24.1.2 El nuevo paradigma

WebRTC trae un nuevo paradigma para la Web. La API proporciona la tecnología para hacer realidad las aplicaciones de red entre pares para la web. El proceso utiliza un servidor de señalización que establece la conexión, pero la información es intercambiada entre los navegadores de los usuarios sin necesidad de otra intervención, tal como se muestra en la Figura 24-2:

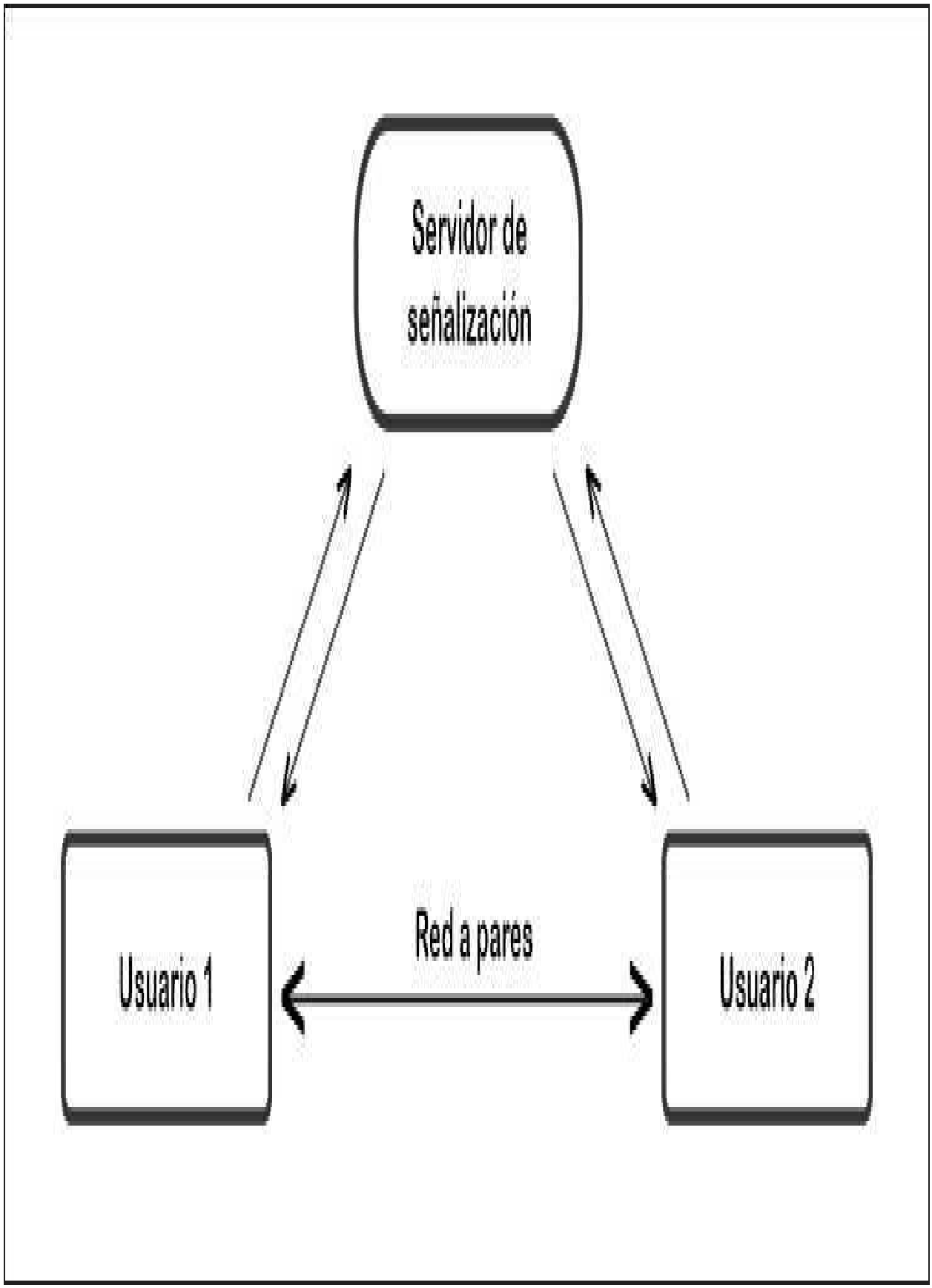

Servidor de señalización
Usuario 1
Red a pares
Usuario 2

Figura 24-2

Nuevo paradigma.

Los servidores aún son necesarios, pero ya no son proveedores de los contenidos. Ahora los servidores envían señales para iniciar la conexión y el contenido se transmite directamente desde el ordenador de un usuario a otro. Gracias a esta tecnología, el streaming de vídeo y de audio, las vídeollamadas, la mensajería instantánea y el intercambio de datos son ahora características disponibles para las aplicaciones que se ejecutan en el navegador.

24.1.3 Requisitos

Sin embargo, hay un problema. En el viejo paradigma, los servidores son de acceso público. Se les asigna una dirección IP (o varias) que los identifican, y los dominios que traducen esas IP hacen que el acceso sea aún más fácil. Los servidores no solo son fáciles de encontrar sino que además son accesibles en todo momento y desde cualquier lugar. Si un servidor cambia la IP, su dominio se traslada a la nueva IP casi instantáneamente (se toma un par de horas en propagar la nueva información a través de la red, pero el cambio es instantáneo). Los ordenadores de los usuarios, por otro lado, no pueden tener una dirección IP única. Tienen IP privadas que luego son traducidas a IP públicas por un sistema llamado NAT, del inglés Network Address Translator (Traductor de dirección de red). Este sistema establece rutas hacia los ordenadores de los usuarios que suelen ser difíles de seguir y varían en cada caso. Como si esto no fuera suficientemente complicado, los equipos de los usuarios también se ocultan detrás de un cortafuegos, tienen diferentes puertos a través de los cuales son transferidos los datos o incluso llegan a ser inaccesibles sin una previa advertencia. Conectar a un usuario con otro requiere de información que los navegadores son incapaces de proporcionar. Necesitan ayuda.

Teniendo en cuenta esta situación, la API WebRTC fue desarrollada para trabajar junto con los servidores que controlan y devuelven la información necesaria para que la aplicación pueda acceder al ordenador del usuario. Estos servidores funcionan bajo una estructura llamada ICE, acrónimo de Interactive Connectivity Establishment (Establecimiento interactivo de conectividad), que permite encontrar la mejor manera de conectar un usuario con otro. ICE es el nombre de un proceso que obtiene la información a través de servidores y sistemas diferentes. La Figura 24-3 muestra cómo se establece la nueva estructura.

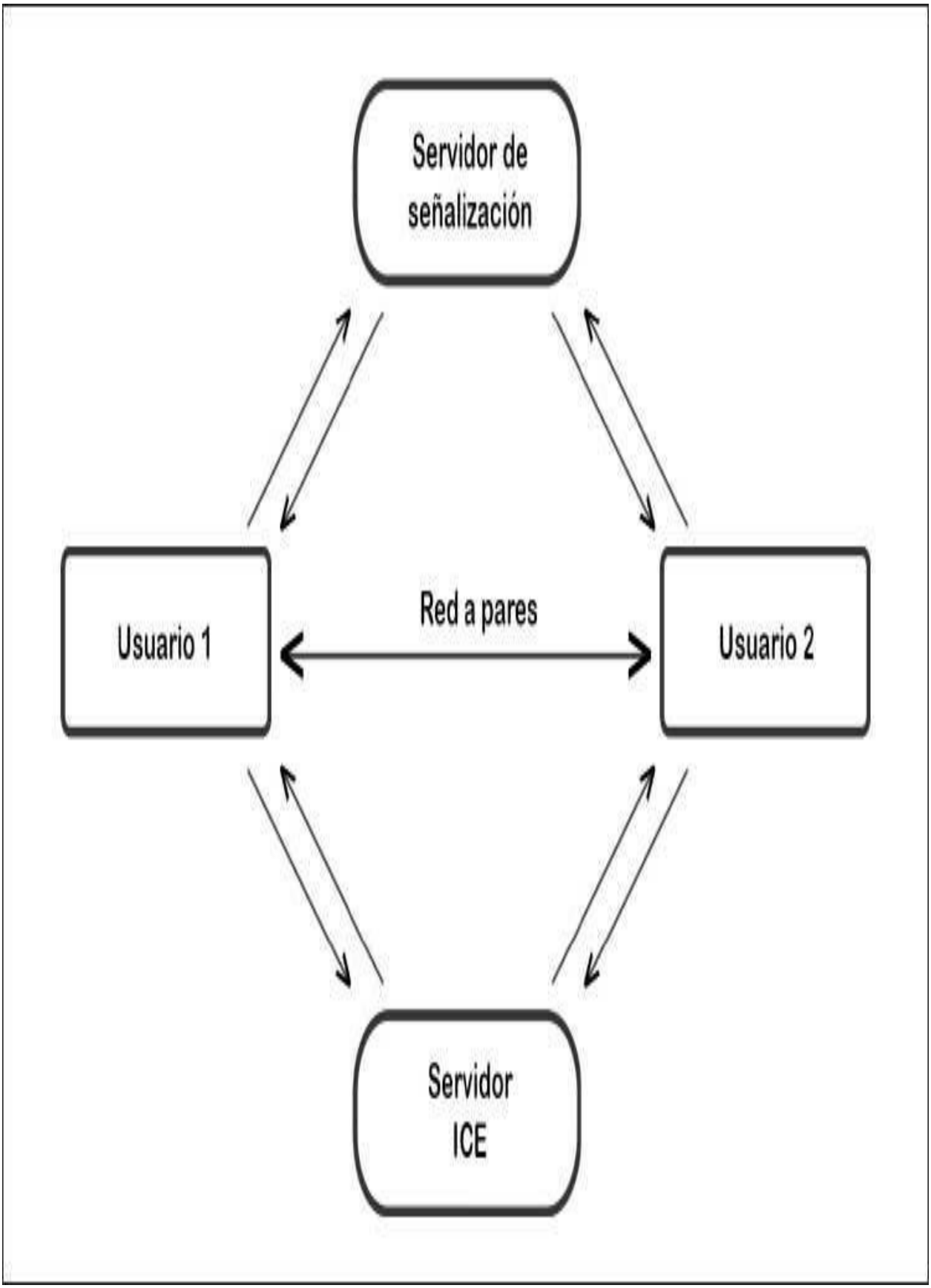

Servidor de señalización
Usuario 1
Red a pares
Usuario 2
Servidor ICE

Figura 24-3

Incluir servidores ICE.

Los servidores ICE son por lo general servidores STUN y TURN. Un servidor STUN descubre y devuelve una dirección IP pública del equipo del usuario y también información acerca de cómo el NAT fue configurado para ese equipo. Un servidor TURN proporciona una conexión usando una dirección IP en la nube cuando las otras alternativas no están disponibles. Los sistemas en ambos extremos del proceso de comunicación permiten decidir qué conexión es más eficiente y proceder a través de esa ruta.

La estructura descrita en la Figura 24-3 es la siguiente: los navegadores de los usuarios 1 y 2 de acceso a los servidores del ICE para obtener la información que describe cómo cada equipo se ve en Internet. Ambas aplicaciones pueden acceder al servidor de señalización y enviar sus descripciones al otro extremo. Una vez que la información llega y ambas partes están de acuerdo sobre el camino a seguir, la conexión de red entre pares se ha establecido, y los usuarios se conectan sin necesidad de intervención de los servidores más (a menos que estén desconectados y la conexión se deba establecer de nuevo).

24.1.4 RTCPeerConnection

El primer paso para crear la conexión entre pares es proporcionar el navegador la información acerca de la conexión, los datos que deben ser compartidos y los servidores de ICE que serán usados. Esto se hace mediante las propiedades, métodos y eventos proporcionados por el objeto RTCPeerConnection. La API proporciona el siguiente constructor para obtener este objeto:

RTCPeerConnection(configuración): Este pimer constructor devuelve un objeto RTCPeerConnection. El objeto se utiliza para proporcionar al navegador la información necesaria para establecer una conexión de red entre pares. El atributo configuración es un objeto con información para encontrar los servidores ICE.

24.1.5 Candidato ICE

Cuando el proceso ICE encuentra una manera de comunicarse con un ordenador, la información recuperada es llamada candidato ICE.

Los candidatos son establecidos y añadidos a la conexión de red de pares usando los siguientes elementos:

RTCIceCandidate(candidateInitDict): Este constructor devuelve un objeto RTCIceCandidate. El atributo candidateInitDict proporciona la información para inicializar el objeto (normalmente la información enviada por el par remoto).

addIceCandidate(candidate): Este método agrega un candidato ICE a distancia para la conexión entre pares. El atributo candidate es un objeto devuelto por el constructor RTCIceCandidate().

iceState: Esta propiedad devuelve el estado del proceso ICE para la conexión del par actual.

24.1.6 Oferta y respuesta

La comunicación entre pares se inicia por dos procesos llamados oferta (offer) y respuesta (answer) Cuando una aplicación desea iniciar una conexión, envía una oferta a la aplicación del otro lado a través del servidor de señalización. Si esta oferta es aceptada, la segunda aplicación devuelve una respuesta. Estos son los métodos proporcionados por la API para este propósito:

createOffer(success, error): Este método se utiliza para crear una oferta. Se genera un blob que contiene la descripción del equipo local (llamada SessionDescription). La descripción contiene los flujos de medios, códecs negociados para la sesión, los candidatos ICE y también una propiedad que describe su tipo (en este caso, offer). La SessionDescription obtenida por este método se envía a la función success para que pueda ser procesada. Esta función se encarga de enviar la oferta al par remoto.

createAnswer(success, error): Este método se utiliza para crear una respuesta. Genera un blob que contiene la descripción del equipo local (llamada SessionDescription). Esta descripción contiene los flujos de medios, códecs negociados de la sesión, los candidatos del ICE y también una propiedad que describe su tipo (en este caso, answer). La SessionDescription obtenida por este método se envía a la función success para que pueda ser procesada. Esta función se encarga de enviar la respuesta al par remoto.

24.1.7 SessionDescription

Como hemos comentado, la descripción de cada equipo, incluyendo los flujos de medios, códecs negociados de la sesión y los candidatos ICE, se llama SessionDescription. Las SessionDescriptions son enviadas de un usuario a otro a través de servidores de señalización y luego declaradas para la conexión de red entre pares con los siguientes métodos:

setLocalDescription(description, success, error): Este método proporciona la descripción del equipo local para la conexión de red entre pares. El atributo description es el objeto devuelto por los métodos createOffer() y createAnswer(). Los atributos success y error son las funciones que serán llamadas en caso de éxito o fracaso.

setRemoteDescription(description): Este método proporciona la descripción del equipo remoto de la conexión de pares. El atributo description es un objeto devuelto por el constructor RTCSessionDescription() y la información recibida por el par remoto.

24.1.8 Flujos de medios o streams

Básicamente, una conexión de red entre pares está hecha para compartir flujos de medios y datos. El proceso para la transmisión de datos implica la creación de canales de datos, como veremos más adelante, pero para agregar o eliminar flujos de datos multimedia a una conexión de red entre pares, solo tenemos que aplicar los siguientes métodos:

addStream(stream): Este método agrega un flujo de medios para la conexión de pares. El atributo stream es una referencia al flujo de medios (devuelta mediante métodos tales como getUserMedia(), por ejemplo).

removeStream(stream): Este método elimina una secuencia de medios de la conexión de red entre pares. El atributo stream es una referencia a la secuencia de medios (devuelta por métodos tales como getUserMedia(), por ejemplo).

24.1.9 Eventos

El proceso de establecer la conexión de red entre pares y obtener información de un ordenador u otro es asíncrona. Una vez que el objeto RTCPeerConnection ha sido creado, tenemos que detectar los eventos y esperar los resultados. Aquí está una lista de los eventos disponibles:

negotiationneeded: Este evento se activa si es necesaria una nueva sesión de negociación (por ejemplo, si debe hacerse una nueva oferta).

icecandidate: Este evento se activa cuando está disponible un nuevo candidato ICE.

stateChange: Este evento se activa cuando el estado de la conexión de red entre pares cambia.

addstream: Este evento se activa cuando el par remoto agrega un flujo de medios.

removestream: Este evento se activa cuando el par remoto elimina un flujo de medios.

gatheringchange: Este evento se activa cuando el estado del proceso ICE cambia.

icechange: Este evento se activa cuando el estado del ICE cambia (por ejemplo, porque se ha establecido un nuevo servidor).

datachannel: Este evento se activa cuando el par remoto agrega un canal de datos nuevo.

24.1.10 El final

Los usuarios tienen que tener la capacidad de iniciar y finalizar una conexión de red entre pares. Para cerrar la conexión, la API proporciona el siguiente método:

close(): Este método cambia el estado del objeto RTCPeerConnection a closed y acaba cualquier tipo de flujo de medios y procesamiento ICE que pudiera estar activo, cerrando así la conexión.

24.2 Ejecutar WebRTC

Como explicamos en la introducción de este capítulo, la estructura necesaria para generar una conexión de red entre pares en la Web implica no solo una aplicación Javascript en cada extremo, sino también servidores que coordinen la conexión. La API deja la configuración del proceso, las tecnologías a utilizar, así como el tipo de servidor y red en manos de los desarrolladores. Así que hay mucho que hacer, además de programar la aplicación. Vamos a empezar con el servidor de señalización.

Importante

Los ejemplos de este capítulo se basan en un procedimiento que consideramos apropiado para las circunstancias de este manual, pero la forma en la que se realiza la conexión, se hace la señalización y se identifican los usuarios puede cambiar completamente en diferentes contextos. Usted tiene que decidir si el uso de este procedimiento es adecuado o no para su aplicación.

24.2.1 Servidor de señalización

Un servidor de señalización no es lo mismo que un servidor de contenido. Los servidores de señalización tienen que establecer conexiones persistentes para poder informar a la aplicación cuando sea recibida una oferta o una respuesta. Como ejemplo, echemos un vistazo a la Figura 24-3 de nuevo. Cuando el Usuario 1 quiere conectarse con el Usuario 2, la aplicación de este equipo tiene que enviar una oferta al servidor de señalización que solicita la conexión, y el servidor tiene que ser capaz de informar al Usuario 2 de que se ha hecho una petición. Este proceso solo es posible si previamente el servidor y los usuarios han establecido una conexión persistente. Por tanto, un servidor de señalización no solo tiene el propósito de recibir y enviar señales (ofertas y respuestas), sino que también tiene que establecer una conexión persistente antes de la configuración de la conexión de red entre pares para mantener informados a los usuarios (por ejemplo, no es posible realizar una vídeollamada si los usuarios no han sido notificados de una llamada entrante).

El trabajo de un servidor de señalización no se detiene allí. También hay que controlar quién está autorizado a crear una conexión y quién es capaz de conectarse con quién. WebRTC no proporciona un método estándar para hacerlo, sino que deja esto las manos de los desarrolladores. La buena noticia es que hay varias opciones disponibles. Ya hemos estudiado cómo crear conexiones persistentes con HTML5 en el Capítulo 23 utilizando WebSockets. La API WebSocket es una opción, pero no la única. Google ofrece la API Google Channel, una API desarrollada para crear una conexión persistente entre las aplicaciones y los servidores de Google. Y los servidores de código abierto que implementan una tecnología llamada SIP, del inglés Session Initiation Protocol (Protocolo de inicio de sesión) ya están disponibles de forma gratuita.

Para nuestro ejemplo hemos decidido implementar el mismo servidor WebSocket en PHP utilizado en el Capítulo 23. La función onMessage() del

archivo server.php tiene que ser modificada para recibir y enviar señales a la conexión correcta:

```
public function onMessage(IWebSocketConnection $user,
 IWebSocketMessage $msg){
  $thisuser = $user->getId();
  $msg = trim($msg->getData());
  $msgback = WebSocketMessage::create($msg);
 foreach($this->server->getConnections() as $user){
    if($user->getId() != $thisuser){
      $user->sendMessage($msgback);
    }
  }
}
```

Código 24-1

Nueva función PHP para el servidor WebSocket (server.php).

Hágalo usted mismo

Igual que en el Capítulo 23, hemos preparado un archivo con el servidor WebSocket listo para instalar, así como el archivo server.php ya adaptado para la aplicación. Vaya a www.minkbooks.com/content/, descargue el paquete webrtc.rar, y cargue la carpeta ws que está dentro de este paquete en su servidor. Usted puede acceder a su servidor a través de una conexión SSH usando un programa como PuTTY (Disponible en www.chiark.greenend.org.uk/~sgtatham/putty/). También puede entrar en la carpeta ws y escribir el comando php server.php* para ejecutar el servidor WebSocket. Para obtener más información, consulte el Capítulo 23.

* Nota del traductor: Inicie una sesión del Símbolo del sistema (que puede ubicar con el buscardor de Windows, por ejemplo; cambie el directorio escribiendo el siguiente comando: cd RUTA, donde RUTA es la ruta al archivo server.php, y pulse la tecla Enter para ejecutar el cambio. Luego introduzca el comando php server.php para ejecutar el servidor WebSocket.

El servidor de nuestro ejemplo no controla nada. Sigue el proceso más básico posible para ayudar a establecer una conexión de red entre pares: toma el mensaje recibido de un usuario y lo envía al otro. No es útil para una aplicación de la vida real pero es suficiente para probar nuestra pequeña aplicación y

entender cómo funciona el proceso.

24.2.2 Servidores ICE

Los servidores ICE se especifican durante la construcción de la conexión de red entre pares. La API proporciona una propiedad para declarar una matriz con la configuración de los servidores ICE para el constructor RTCPeerConnection():

iceServers: Esta propiedad se utiliza para especificar los servidores STUN y TURN disponibles para ser utilizados por el proceso ICE.

El valor de la propiedad iceServers se define como una matriz de objetos que contiene las siguientes propiedades:

url: Esta propiedad declara la dirección URL del servidor STUN o TURN.

credential: Esta propiedad se utiliza para definir una credencial para un servidor TURN.

La sintaxis para introducir esta información es la siguiente: {“iceServers”: [{“url”: “stun: stun.DOMINIO.com:12345”}]}, donde stun.DOMINIO.com es el dominio de un servidor STUN y 12345 es el puerto en el que el servidor está disponible.

Importante

Tenemos que proporcionar nuestros propios servidores ICE para trabajar con nuestra aplicación. La creación y configuración de servidores STUN y TURN está más allá del alcance de este libro. En los ejemplos de este capítulo trabajaremos con un servidor STUN proporcionado por Google para realizar pruebas. Para obtener más información sobre los servidores ICE, visite nuestro sitio web y siga los enlaces de este capítulo.

24.2.3 Documento HTML

Es el momento de crear nuestra aplicación. El uso más común de una conexión de red entre pares es generar una videollamada. Vamos a crear un documento HTML con dos elementos <video> para mostrar el vídeo de la cámara local en una pequeña caja en el lado izquierdo de la pantalla y el vídeo de la cámara remota (la persona que llamamos) en una caja más grande a la derecha. El documento también incluye un botón con el texto Llamar para hacer llamadas y los estilos CSS de las cajas.

```
<!DOCTYPE html>
<html>
<head>
  <title>API WebRTC</title>
  <style>
    #emisor, #receptor{
      float: left;
      margin: 5px;
      padding: 5px;
      border: 1px solid #000000;
    }
  </style>
  <script src="webrtc.js"></script>
</head>
<body>
  <section id="emisor">
    <video id="videoemisor" width="300" height="150"></video>
    <br><input type="button" id="botonllamada" value="Llamada">
  </section>
  <section id="receptor">
    <video id="videoreceptor" width="500" height="500"></video>
  </section>
</body>
</html>
```

Código 24-2

Documento HTML para hacer videollamadas.

El código incluido en el archivo webrtc.js toma el vídeo y audio de la cámara local y del micrófono, los asigna al elemento <video> con la Identificación videoemisor y envía el stream al otro usuario. Cuando llega un flujo de medios desde el otro extremo de la línea, la secuencia de comandos asigna este flujo al segundo elemento <video> identificado como videoreceptor, y finalmente se establece la comunicación.

24.2.4 El código Javascript

El código Javascript de este ejemplo tiene que establecer la conexión de red entre pares, comunicarse con los servidores ICE, obtener la SessionDescription, enviar la oferta y la respuesta, y añadir tanto el stream remoto como el local a la conexión. Pero vamos a hacerlo paso a paso. En primer lugar, tenemos que crear la conexión permanente con el servidor de señalización (en este caso, un servidor WebSocket) y luego capturar el flujo de vídeo desde la cámara web local y el flujo de audio desde el micrófono local.

```
var usuario, socket;
function iniciar(){
  var botonllamada = document.getElementById('botonllamada');
  botonllamada.addEventListener('click', llamar);
  socket = new WebSocket("ws://TU_IP:12345/ws/server.php");
  socket.addEventListener('message', recibir);
  navigator.webkitGetUserMedia({video: true, audio: true},
                               establecercamara, mostrarerror);
}
```

Código 24-3

Conectar con el servidor WebSocket y obtener flujos de medios.

En la función iniciar() del Código 24-3 empezamos añadiendo la función llamar() como detector para el evento clic. Cada vez que se hace clic en el botón Llamar, se ejecuta esta función para enviar una oferta e iniciar la llamada.

A continuación, la aplicación crea una conexión persistente con el servidor WebSocket y añade un detector para el evento message que detecta los mensajes procedentes del servidor de señalización. Esto es útil de ambos lados de la línea: el receptor sabrá cuando está siendo llamado y la persona que llama podrá escuchar la respuesta.

Finalmente, los flujos de medios de la cámara web local y micrófono son capturados por el método getUserMedia() (véase el Capítulo 9). En caso de éxito se llama a la función establecercamara() y, en caso de error, se ejecuta la función mostrarerror(), por ejemplo, si el acceso a la cámara web fue denegado.

```
function establecercamara(stream){
  var video = document.getElementById("videoemisor");
  video.setAttribute('src', URL.createObjectURL(stream));
  video.play();

  var servers = {"iceServers": [{"url": "stun: stun.l.google.com:
      19302"}]};
  usuario = new webkitRTCPeerConnection(servers);
  usuario.addStream(stream);
  usuario.addEventListener('addstream', streamremoto);
  usuario.addEventListener('icecandidate', estableceric);
}
function mostrarerror(){
  console.log('Error');
}
```

Código 24-4

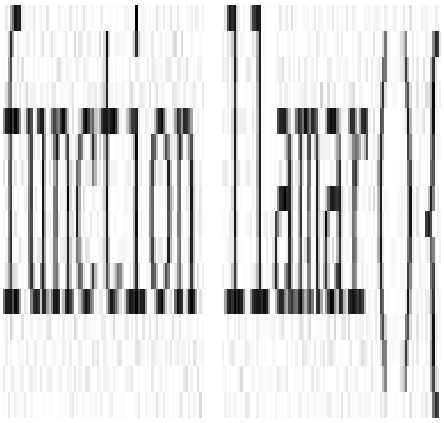
Inicio de la conexión entre pares.

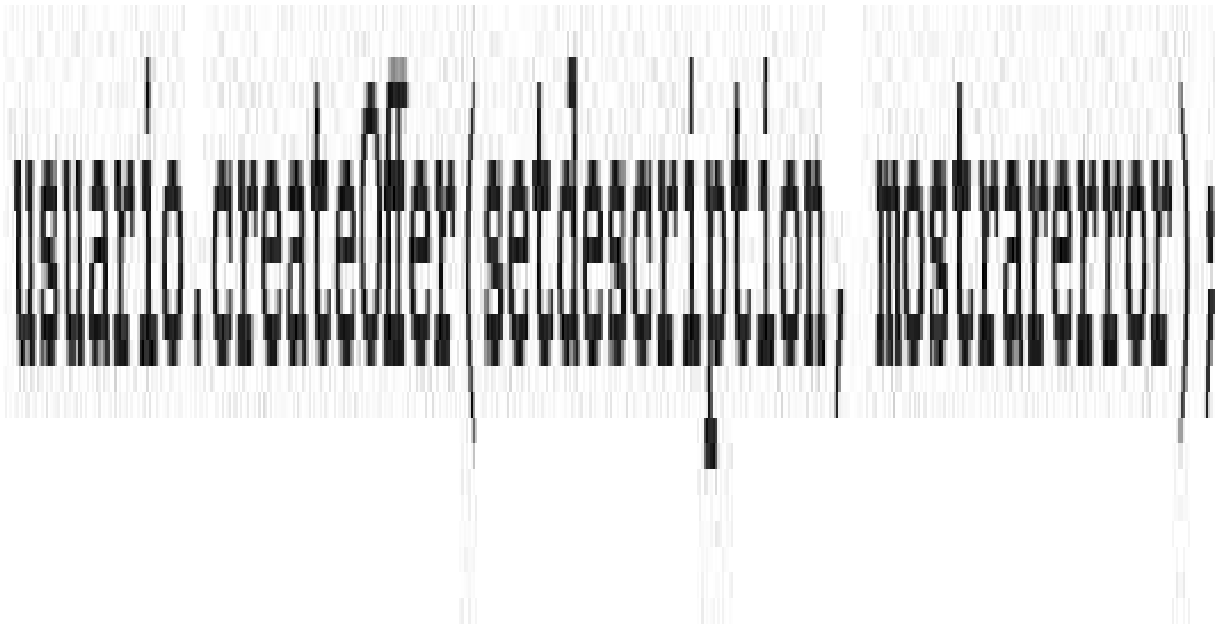
La función mostrarerror() será llamada por varios métodos, así que solo la utilizaremos para mostrar un mensaje de error en la consola. Sin embargo, la función establecercamara() tiene que hacer más. Debe establecer la conexión de red entre pares, y asignar los flujos de medios de la cámara web y el micrófono al elemento <video>.

Al principio de la función establecercamara() en el Código 24-4, las secuencias de vídeo y audio capturadas por el método getUserMedia() se asignan al elemento <video> correspondiente (el vídeo pequeño de la izquierda de la pantalla) y, luego éste es ejecutado gracias al método play(). La conexión de pares se inicia a continuación, al declarar los servidores ICE disponibles y crearse el objeto RTCPeerConnection con esta información.

El servidor STUN proporcionado por Google se establece como nuestro servidor ICE en este ejemplo. El objeto RTCPeerConnection proporciona al navegador la información necesaria para establecer la conexión. Parte de esta información está disponible de inmediato, como el flujo de medios locales, pero el resto del proceso es asíncrono, por lo que tenemos que detectar los eventos para capturar esta información cuando está disponible. Por esta razón, después de añadir el vídeo local al objeto mediante el método addStream(), añadimos detectores a los eventos addstream e icecandidate. El evento addstream se activará cuando el par remoto active su cámara y el evento icecandidate se disparará cuando el par local establezca un candidato ICE.

Para que todo esto suceda, primero hay que establecer la conexión de red entre pares. Ésta es la función que utilizaremos para crear la oferta e iniciar la llamada:

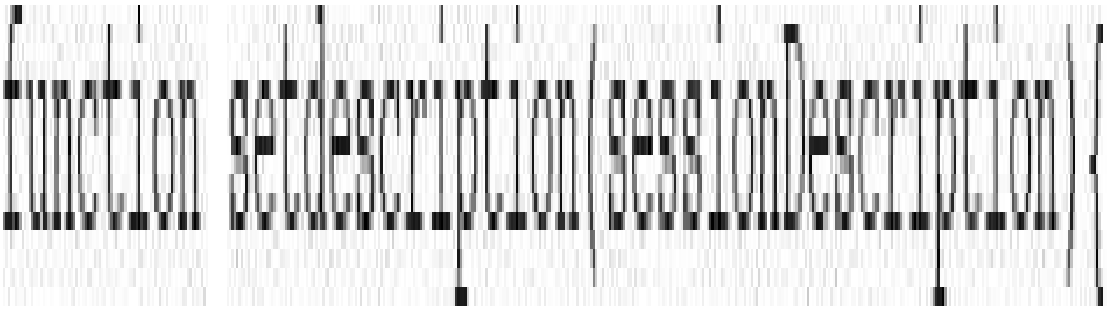


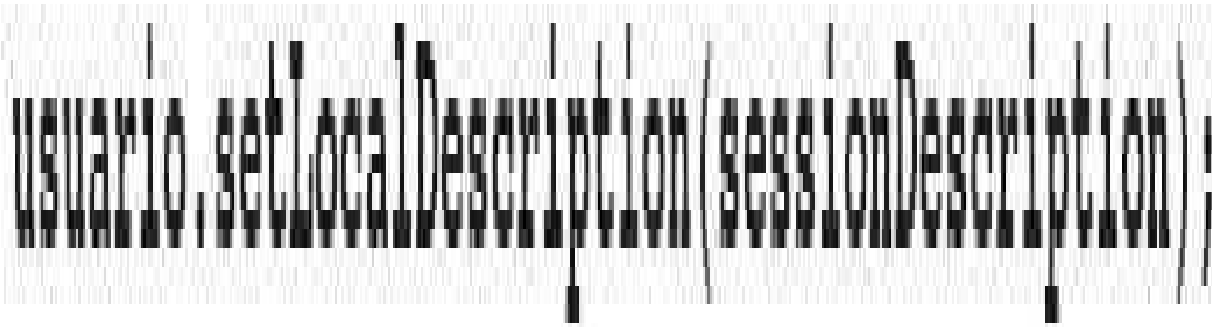


Código 24-5

Obtener la SessionDescription.

Cuando el vídeo y el audio de la cámara web y del micrófono locales se estén reproduciendo en la pantalla en ambos puntos, estaremos listos para realizar la llamada. Al hacer clic en el botón Llamar se ejecuta la función llamar(), que genera la descripción del equipo local del tipo de oferta y, en caso de éxito, llama a la función SetDescription() con esta información. La función SetDescription() recibe la descripción del equipo, establece esta información como la descripción local y la envía al interlocutor remoto para iniciar la conexión.


function setDescription(sessionDescription){


vguario.setLocalDescription(sessionDescription);

enviarmensaje(sessionDescription);

Código 24-6

Enviar una oferta a un cliente remoto.

El método setLocalDescription() es utilizado en la función del Código 24-6 para proporcionar al navegador la información que describe el par local. Esta información es enviada por la función enviarmensaje() al par remoto y se encarga de enviar mensajes al servidor WebSocket poder establecer la conexión:

```
function enviarmensaje(message){

    var msg = JSON.stringify(message);

    socket.send(msg);

}
```

Código 24-7

Enviar de señales al par remoto.

El método send() de la API WebSocket tiene sus limitaciones. Por esta razón, antes de enviar los objetos Javascript al servidor, tenemos que convertirlos en cadenas JSON. En la función del Código 24-7 hacemos esto con el método JSON. stringify().

Conceptos básicos

JSON, acrónimo de Javascript Object Notation (Notación del objeto Javascript), es un formato de datos desarrollado específicamente para compartir datos. Un valor JSON es un texto con un formato específico que puede ser enviado como una cadena de texto normal pero transformado en objetos útiles por casi cualquier lenguaje de programación. Javascript proporciona dos métodos para trabajar con JSON: JSON.stringify() para convertir los objetos Javascript en cadenas JSON y JSON.parse() para convertir cadenas JSON en objetos Javascript.

La función enviarmensaje() está a cargo del proceso de señalización. Cada vez que uno de los pares hace una oferta, envía una respuesta o comparte candidatos ICE, la información se transmite al servidor WebSocket través de mensajes enviados por esta función. El formato de estos mensajes y la forma en la que estas señales se envían y procesan depende del desarrollador. Para nuestra aplicaciónhemos utilizado la propiedad type enviada las descripciones del

equipo para comprobar el tipo de mensaje recibido. Esta función recibe y procesa las señales:

```
function recibir(e){
  var msg = JSON.parse(e.data);
  switch(msg.type){
    case 'offer':
      usuario.setRemoteDescription(new RTCSessionDescription(msg));
      usuario.createAnswer(setdescription, mostrarerror);
      break;
    case 'answer':
      usuario.setRemoteDescription(new RTCSessionDescription(msg));
      break;
    case 'candidate':
      var candidate = new RTCIceCandidate(msg.candidate);
      usuario.addIceCandidate(candidate);
  }
}
```

Código 24-8

Procesar las señales.

La función recibir() del Código 24-8 es llamada cada vez que se dispara el evento message del objeto WebSocket. Esto significa que cada vez que el servidor WebSocket envía un mensaje a la aplicación, esta función es ejecutada. Aquí comprobamos el valor de la propiedad type en cada mensaje de señal y seguimos el procedimiento de acuerdo a cada caso. Si el valor de la propiedad type es offer, significa que la aplicación ha recibido una oferta de un usuario que llama. En este caso tenemos que establecer la descripción del equipo del interlocutor remoto y enviar una respuesta con el método createAnswer(). Sin embargo, cuando el mensaje de la señal es del tipo answer, lo que significa que la llamada ha sido aceptada, se establece la descripción del interlocutor remoto y se establece la conexión. El último caso de la declaración switch()comprueba si el mensaje de señal es del tipo candidate. En este caso, el candidato ICE enviado por el par remoto se añade al objeto por el objeto RTCPeerConnection mediante el método addIceCandidate().

Importante

El procedimiento acabamos de describir es el que hemos utilizado para este ejemplo. La API WebRTC no define ningún procedimiento estándar para procesar los mensajes de señalización o para enviarlos (también el uso del servidor de WebSocket fue nuestra elección). Usted debe decidir si la aplicación de este procedimiento es adecuada para su aplicación o no.

Cuando la oferta es aceptada y la respuesta es recibida, ambos equipos comparten los vídeos capturados por las respectivas cámaras así como la información sobre los servidores ICE que se utilizarán para establecer la conexión. Este proceso se dispara los eventos addstream y icecandidate, ejecutando las funciones streamremoto() y la estableceric()en ambos extremos:

```
function streamremoto(e){
  var video = document.getElementById("videoreceptor");
  video.setAttribute('src', URL.createObjectURL(e.stream));
  video.play();
}

function estableceric(e){
  if (e.candidate) {
    var message = {
      type: 'candidate',
      candidate: e.candidate,
    };
    enviarmensaje(message);
  }
}
```

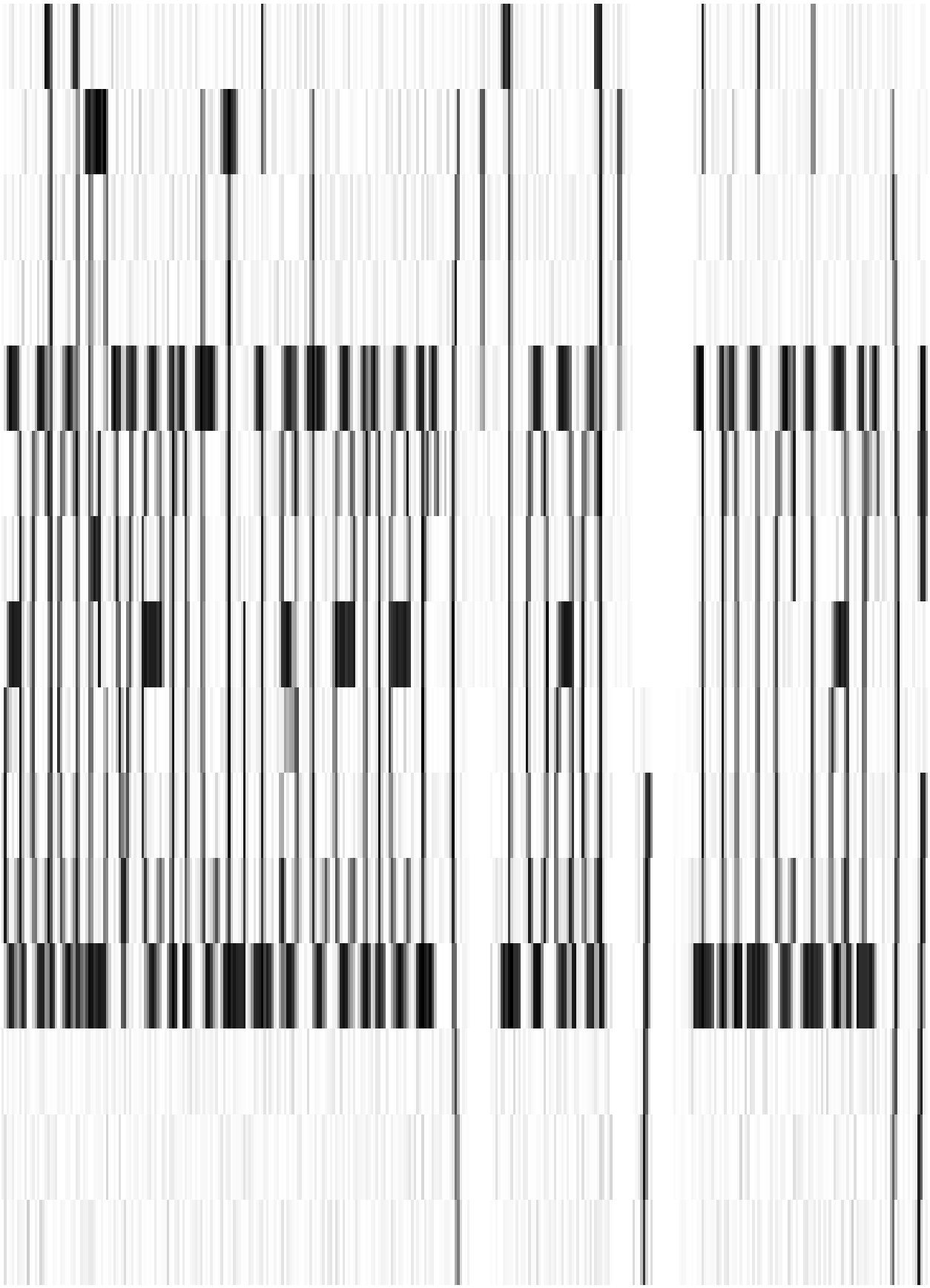
Código 24-9

Responder a los eventos addstream y icecandidate.

Tan pronto como es creado el objeto RTCPeerConnection, el navegador comienza a negociar con los servidores ICE para obtener la información necesaria para acceder al ordenador local. Cuando esta información es finalmente recuperada, la aplicación es informada por el evento icecandidate. En la función estableceric() del Código 24-9, la información proporcionada por el evento es usada para generar un mensaje de señalización del tipo candidate y enviarlo al par remoto. Nótese que en esta ocasión hemos tenido que declarar explícitamente la propiedad type porque el objeto candidate no lo incluye. Deberá seguir este procedimiento para ser coherente con el resto del proceso de señalización creado anteriormente en la función recibir().

Después de que los pares acuerdan la ruta a seguir para la conexión, son compartidos los flujos de medios. Si uno de los pares inicia un nuevo flujo de medios, se le informa al otro a través del evento addstream. Cuando este evento se dispara, simplemente se asigna el flujo de medios remoto a un elemento <video>, del mismo modo que se hizo antes para el flujo de medios locales de la cámara web y el micrófono. La función streamremoto() del Código 24-9 crea la URL correspondiente para señalar el flujo de medios remoto y asignarlo como fuente del elemento <video> identificado como videoreceptor. Esto se realiza en ambos puntos de la conexión, por lo que cada usuario tendrá su propia imagen en el lado izquierdo de la pantalla y la imagen de la otra persona en el lado derecho.

Lo último que debemos hacer para que nuestra aplicación esté lista, es detectar el evento load y ejecutar la función función iniciar() tan pronto como se haya cargado el documento:



Código 24-10

Escuchar el evento load para iniciar la aplicación.

Hágalo usted mismo

Copie el Código 24-2 en un documento HTML. Cree un archivo llamado webrtc.js y copie en él las declaraciones presentadas del Código 24-3 al Código 24-10. Suba los dos archivos a su servidor, ejecute el servidor WebSocket siguiendo el procedimiento que hemos explicado antes y abra el documento en su navegador. Para conectar otra persona tendrá para abrir el documento en dos equipos diferentes, pero para probar la aplicación puede abrirla en ventanas diferentes del mismo equipo.

Importante

Debe cambiar la cadena TU_IP por la IP de su servidor o, si trabaja en un servidor local, la cadena localhost. Para propósitos de prueba puede utilizar su dominio, pero en una aplicación profesional use la IP para evitar el proceso de traducción DNS.

24.2.5 Aplicación de la vida real

WebRTC es una API muy especial. Proporciona una característica muy poderosa que cambiará la web y además deja los procesos críticos en manos del desarrollador. Permite controlar todo: si la conexión ya está establecida o no, si el usuario está autorizado para realizar la llamada, la forma en la que los usuarios escucharán las posibles llamadas, cómo van a ser informados de las llamadas entrantes, etc. El ejemplo que hemos estudiado es lo más sencillo posible. Llamamos a los métodos básicos y creamos un sencillo proceso que simplemente establece la conexión. Debido a la naturaleza de esta API, la forma en la que se convierte en una aplicación real depende totalmente de ti y del tiempo del que dispones para prepararla.

2.4.3 Canales de Datos

La característica más revolucionaria de la API con la que estamos trabajando no es la transmisión de flujos de medios sino la transmisión de datos, que se realiza a través de canales de datos. Estos canales son creados mediante una conexión de pares existente y permiten a los usuarios compartir cualquier tipo de datos, que pueden ser posteriormente convertidos en archivos o contenidos en el otro par.

24.3.1 Creación de canales de datos

La API WebRTC proporciona los métodos siguientes para crear canales de datos:

createDataChannel(etiqueta, configuración): Este método devuelve un objeto RTCDataChannel que representa un canal de datos. Los atributos etiqueta y configuración identifican el canal y un objeto que proporciona los parámetros de configuración, respectivamente. El par remoto es informado de la creación de la canal por el evento datachannel mencionado antes.

El objeto RTCDataChannel incluye las siguientes propiedades, métodos y eventos para configurar y trabajar con canales de datos:

label: Esta propiedad devuelve la etiqueta que fue asignada al canal cuando se creó.

reliable: Esta propiedad devuelve el valor true si el canal fue establecido como confiable al crearlo, o false en caso contrario.

readyState: Esta propiedad devuelve el estado del canal.

bufferedAmount: Esta propiedad nos permite conocer los datos solicitados que aún no han sido enviados al interlocutor remoto. El valor devuelto puede ser utilizado para regular la cantidad de datos y la frecuencia de las

solicitudes con el fin de no saturar la conexión.

binaryType: Esta propiedad define la forma en la que se van a mostrar los datos binarios en el código. Puede tomar dos valores: blob o arraybuffer.

send(datos): Este método envía el valor del atributo datos al par remoto. Los datos se deben especificar como una cadena de texto, un blob o un arrayBuffer.

close(): Este método cierra el canal de datos.

mesage: Este evento se activa cuando se recibe un nuevo mensaje a través de un canal de datos, es decir, cuando el par remoto envía nuevos datos.

open: Este evento se activa cuando el canal es abierto close: Este evento se activa cuando el canal es cerrado.

error: Este evento se activa cuando se produce un error.

2.4.4. Envío de datos

Para demostrar cómo funcionan los canales de datos, vamos a añadir una sala de chat debajo de los videos para que los usuarios puedan enviarse mensajes mientras hablan. El nuevo documento HTML incluye, para este fin, dos cajas pequeñas en la parte superior de los videos, un campo de entrada y un botón para escribir y enviar los mensajes, y una caja para mostrar la conversación bajo los vídeos.

```
<!DOCTYPE html>
<html>
<head>
  <title>WebRTC API</title>
  <style>
   #emisor, #receptor{
   float: left;
   margin: 5px;
   padding: 5px;
   border: 1px solid #000000;
  }
#botones{
   clear: both;
   width: 528px;
   text-align: center;
  }
#cajadatos{
   width: 526px;
   height: 300px;
   margin: 5px;
   padding: 5px;
   border: 1px solid #000000;
  }
 </style>
 <script src="webrtc-11.js"></script>
</head>
 <body>
  <section id="emisor">
   <video id="videoemisor" width="250" height="200"></video>
  </section>
  <section id="receptor">
   <video id="videoreceptor" width="250" height="200"></video>
  </section>
  <nav id="botones">
   <input type="button" id="botonllamada" value="Llamar">
   <input type="text" id="entrada" size="60" required>
   <input type="button" id="botonenviar" value="Enviar">
  </nav>
  <section id="cajadatos"></section>
</body>
</html>
```

Código 24-11

Documento HTML para probar los canales de datos

El código Javascript para el archivo webrtc.js es similar al ejemplo anterior, pero añade todas las funciones necesarias para crear el canal de datos y transferir los mensajes de un par al otro.

```
var usuario, socket, channel, channelopen;
function iniciar(){
  var botonllamada = document.getElementById('botonllamada');
  botonllamada.addEventListener('click', llamar);
  var botonenviar = document.getElementById('botonenviar');
  botonenviar.addEventListener('click', enviarmensaje);

  socket = new WebSocket("ws://TU_IP:12345/ws/server.php");
  socket.addEventListener('message', recibir);

  navigator.webkitGetUserMedia({video: true, audio: true},
 establecercamara, mostrarerror);
}
function mostrarerror(e){
  console.error(e);
}
function establecercamara(stream){
  var servers = {"iceServers": [{"url": "stun:stun.l.google.
 com:19302"}]};
  usuario = new webkitRTCPeerConnection(servers);
  usuario.addEventListener('addstream', streamremoto);
  usuario.addEventListener('icecandidate', estableceric);
  usuario.ondatachannel = function(e){
    channel = e.channel;
    channel.onmessage = receivemessage;
    channelopen = true;
  };
  usuario.addStream(stream);

  var video = document.getElementById("videoemisor");
  video.setAttribute('src', URL.createObjectURL(stream));
  video.play();
}
function recibir(e){
  var msg = JSON.parse(e.data);
  switch(msg.type){
  case 'offer':
    usuario.setRemoteDescription(new RTCSessionDescription(msg));
    usuario.createAnswer(setdescription, mostrarerror);
    break;
  case 'answer':
    usuario.setRemoteDescription(new RTCSessionDescription(msg));
```

```
    channel = usuario.createDataChannel('data');
    channel.onopen = function(){ channelopen = true; };
    channel.onmessage = receivemessage;
    break;
  case 'candidate':
    var candidate = new RTCIceCandidate(msg.candidate);
    usuario.addIceCandidate(candidate);
  }
}
function sendmsg(message){
    var msg = JSON.stringify(message);
    socket.send(msg);
}
function llamar(){
    usuario.createOffer(setdescription, mostrarerror);
}
function setdescription(sessionDescription){
    usuario.setLocalDescription(sessionDescription);
    sendmsg(sessionDescription);
}
function streamremoto(e){
    var video = document.getElementById("videoreceptor");
    video.setAttribute('src', URL.createObjectURL(e.stream));
    video.play();
}
function estableceric(e){
  if (e.candidate){
    var message = {
      type: 'candidate',
      candidate: e.candidate,
    };
    sendmsg(message);
  }
}
function enviarmensaje(){
  var cajadatos = document.getElementById('cajadatos');
  if(!channelopen){
    cajadatos.innerHTML = 'El canal de datos no est√° listo.
 Int√©ntelo de nuevo';
  }else{
    var message = document.getElementById('entrada').value;
    var chatroom = cajadatos.innerHTML;
    cajadatos.innerHTML = 'Par local dice: ' + message + '<br>';
    cajadatos.innerHTML += chatroom;
    channel.send(message);
  }
}
function receivemessage(e){
  var cajadatos = document.getElementById('cajadatos');
  var chatroom = cajadatos.innerHTML;
  cajadatos.innerHTML = 'Par remoto dice: ' + e.data + '<br>';
  cajadatos.innerHTML += chatroom;
}
addEventListener('load', iniciar);
```

Código 24-12

Creación de conexión de red entre pares con un canal de datos.

El canal de datos tiene que ser creado por uno de los pares mediante el método createDataChannel(). El procedimiento solo puede realizarse después de que la conexión entre pares está listo. Una manera de detectarlo es de esperar por una respuesta. Si no se recibe respuesta, significa que la conexión ha sido aceptada y establecida. Por esta razón hemos creado el canal de datos de este ejemplo en la función recibir(). Cuando un mensaje del tipo answer es recibido desde el servidor de señalización, se crea el canal de datos y asignado a la variable channel. Este proceso es asíncrono: el canal se añade a la conexión de pares y el navegador informa a la aplicación cuando el canal está preparado a través del evento open. Para informar al resto de la aplicación cuando el canal está listo para ser utilizado, detectamos este evento y cambiamos el valor de la variable channelopen a true cuando se activa el evento. También detectamos el evento message para comprobar los mensajes que vienen del otro interlocutor a través del canal.

Cuando un par crea un canal, el otro interlocutor detecta la acción mediante el detector del evento evento datachannel. En la función iniciar() hemos declarado un controlador de este evento. El evento devuelve un objeto con la propiedad channel que proporciona una referencia para el nuevo canal. Esta referencia se almacena en la variable channel; así mismo, se registra un controlador del evento message también de este lado para comprobar los mensajes que llegan a través del canal.

Ahora ambos pares tienen el canal de datos abierto, tienen registrado un controlador para el evento message y están dispuestos a escuchar y enviar mensajes. Dos funciones se han creado para este fin: enviarmensaje() y

receivemessage(). Estas funciones toman los mensajes generados por los pares local y remoto y los insertan en el cuadro cajadatos de cada aplicación. La función sendMessage() se ejecuta cada vez que se hace clic en el botón Enviar. Esta función lee la variable channelopen para comprobar si el canal está abierto, y, si lo está, toma el valor del campo entrada, lo muestra en la pantalla y lo envía al otro interlocutor con el método send(). Cuando el mensaje es recibido por el otro interlocutor, el evento message se activa y es llamada la función receivemessage(). Esta función simplemente toma el valor de la propiedad data del objeto devuelto por el evento y lo muestra en pantalla.

Hágalo usted mismo

Copie el Código 24-11 en un archivo HTML y el Código 24-12 en el archivo webrtc.js. Suba los dos archivos a su servidor, ejecute el servidor WebSocket tal como hemos explicado antes y abra el documento HTML en su navegador. Para conectar con otra persona, tendrá que abrir el documento en equipos diferentes, pero para probar la aplicación que solo se puede abrirla en ventanas diferentes del mismo equipo. Introduzca un mensaje en el campo de entrada y pulse el botón Enviar. Recuerde volver a reemplazar la cadena de texto TU_IP con la IP de su servidor o las palabras localhost, si trabaja con un servidor local.

Importante

Para el momento de escribir este manual, la especificación para canales de datos está aún en proceso de desarrollo y esta parte de la API no ha sido aún implementada en los navegadores. Para probar el ejemplo anterior, se utilizó un polyfill desarrollado por Jesús Leganés Combarro (pirannafs.blogspot.com), disponible en https://github.com/piranna/DataChannel-polyfill. Este código implementa canales de datos utilizando su propio servidor WebSocket. Por esta

razón, el Código 24-12 podría tener que ser modificado para trabajar con la implementación oficial. Por favor, consulte nuestro sitio web para obtener actualizaciones.

25 API Web Audio

25.1 Estructura de nodos de audio

El procesamiento de audio es algo que normalmente no consideramos esencial para la Web. Frente a la posibilidad de crear aplicaciones ricas en gráficos 2D y 3D, el procesamiento de audio llega a parecer irrelevante. ¡Pero esto es un error! No importa cuán versátiles sean los elementos <video> y <audio>, solo el procesamiento de audio puede convertirlos en herramientas profesionales. Si tiene que reproducir un simple video pero no puede ecualizarlo, puede parecer de aficionados. Puede insertar el sonido limpio de una arma de fuego en su juego, pero solamente con variaciones de intensidad podrá hacer que suene como si hubiera sido disparada desde el fondo de la escena 3D. Puede tener grandes sonidos de motores para los coches en su juego de carreras, pero la sensación de que el coche ha pasado solo puede lograrse mediante la aplicación de un efecto Doppler.

HTML5 ofrece a los desarrolladores todas las herramientas para crear completas aplicaciones web. La capacidad de procesar sonidos y cualquier tipo de fuentes de audio no es solo necesaria sino también esencial para este propósito. La API Web Audio es una solución profesional completa que permite el procesamiento de audio para la Web. Se trata de una API de bajo nivel desarrollada por ingenieros de sonido, pero es accesible a cualquier persona. Gracias a ella, los desarrolladores web disponen de las herramientas para desarrollar aplicaciones de producción de audio y motores de audio de juegos que hasta ahora estaban reservadas para ordenadores de escritorio y consolas de juegos.

25.1.1 Los nodos de audio

La API Web Audio utiliza una organización modular en la que cada elemento es un nodo (llamado oficialmente AudioNode). Los nodos representan todas las partes del sistema de audio, desde el sonido en bruto hasta los altavoces. Los nodos están conectados entre sí para formar la estructura final que producirá el sonido.

La Figura 25-1 representa la estructura más básica posible: un nodo para la fuente de audio, que produce el sonido, y un nodo para el destino (altavoces, auriculares o cualquier otro elemento configurado para transmitir el sonido real), que hace el sonido audible. Sin embargo, es posible que tengamos más nodos con fuentes de audio, filtros, controles de volumen, delays, efectos, etc. La estructura puede ser tan compleja como lo requiera cada aplicación.

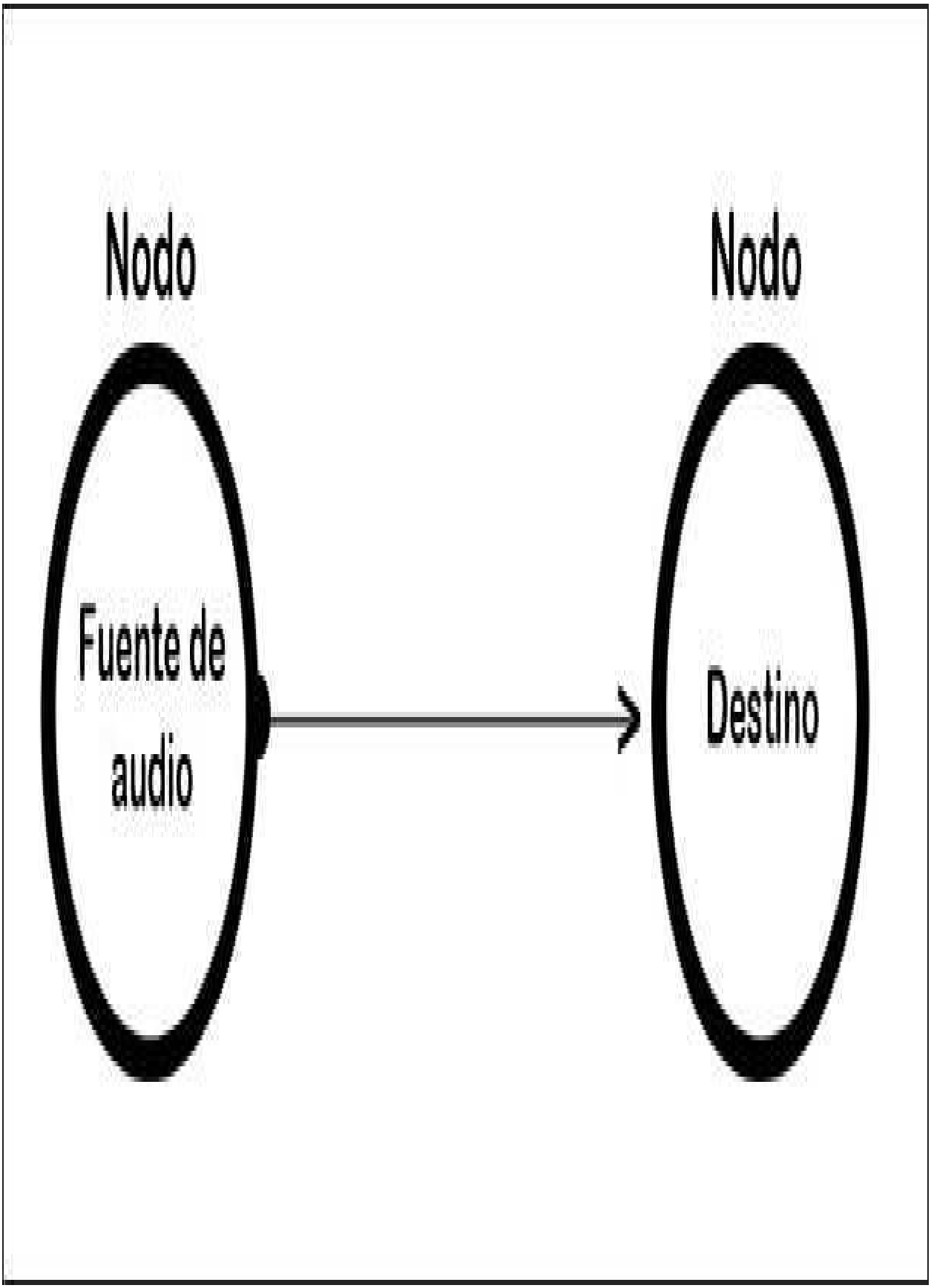

Nodo
Nodo
Fuente de audio
Destino

Figura 25-1

Organización básica de nodos.

El primer nodo de fuente de audio de la Figura 25-2 está conectado al nodo de volumen (GainNode., que a su vez está conectado al nodo compresor (DynamicsCompressorNode., que finalmente está conectado al nodo de destino (altavoces, auriculares, etc.). El volumen de esta fuente de audio será modificado por el nodo de volumen, después será comprimido, automáticamente se mezclará con el resto de los sonidos y finalmente será enviado al nodo de destino y reproducido por la salida de audio del dispositivo (altavoces, auriculares, etc.). Las fuentes de audio 2 y 3 de esta figura van a seguir un camino similar, pero pasarán a través del nodo filtro BiquadFilter. y el nodo de efecto ConvolverNode. antes de alcanzar el nodo de compresor y de destino.

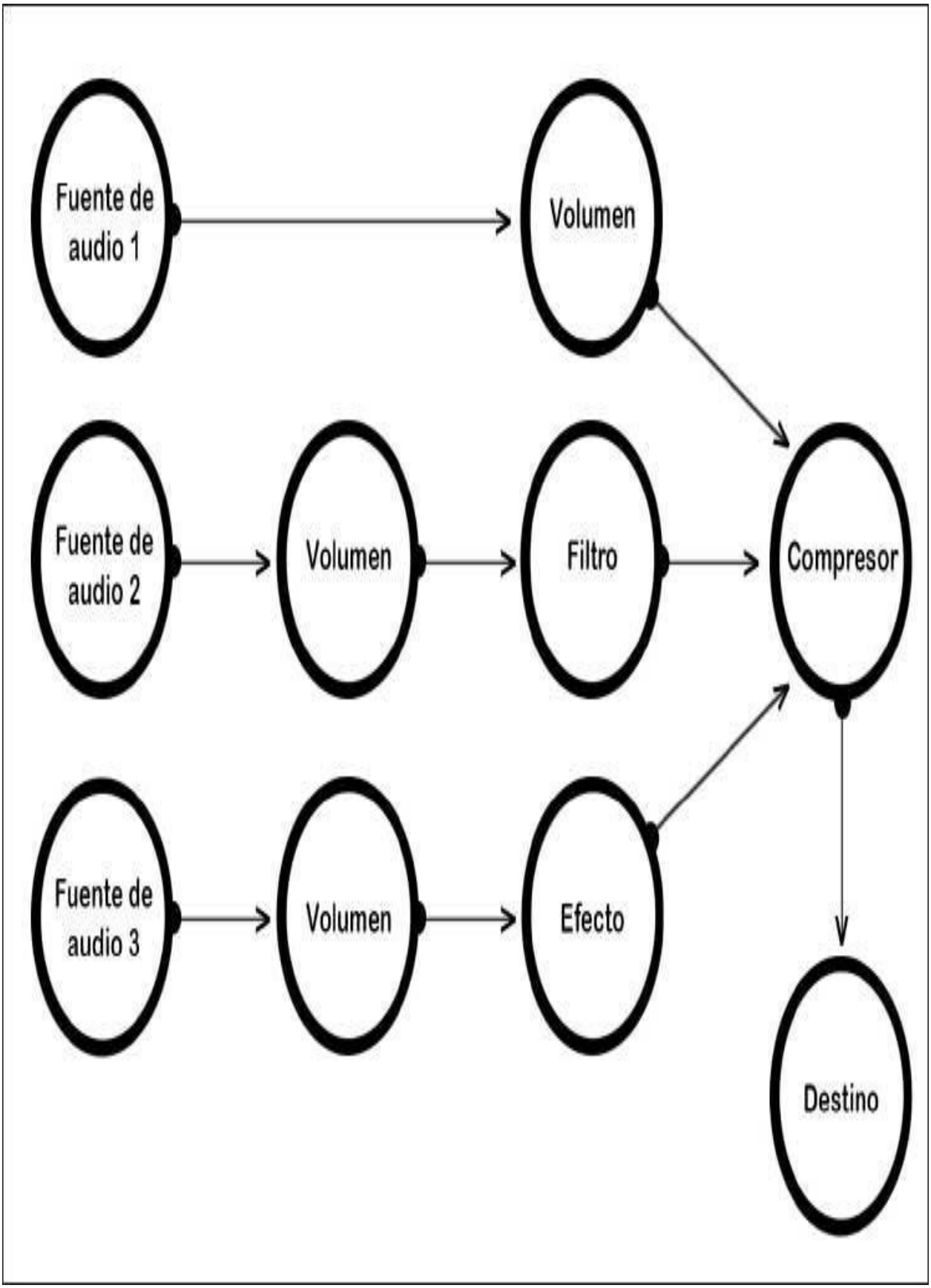

Fuente de audio 1
Volumen
Fuente de audio 2
Volumen
Filtro
Compresor
Fuente de audio 3
Volumen
Efecto
Destino

Figura 25-2

Un sistema de audio más elaborado.

La mayoría de los nodos tienen una conexión de entrada y de salida para recibir sonido desde un nodo anterior, realizar algún tipo de proceso y enviar el resultado al siguiente. Nuestro trabajo es crear los nodos, proporcionar la información que cada uno necesita para funcionar correctamente y conectarlos.

25.1.2 Contexto Audio

Esta estructura de nodos se encuentra en un contexto de audio, que es algo similar al contexto creado por el elemento <canvas>. La API Web Audio ofrece un constructor para obtener el objeto AudioContext:

AudioContext(): Este constructor devuelve un objeto AudioContext. El objeto proporciona métodos para crear cada tipo de nodo disponible, así como algunas propiedades para configurar el contexto y establecer el nodo de destino para acceder a la salida de audio del dispositivo.

Las siguientes son las propiedades proporcionadas por el objeto AudioContext para configurar el contexto y el sistema de audio:

destination: Esta propiedad devuelve un objeto AudioDestinationNode que representa el nodo de destino. Es el último nodo de un sistema de audio y su función es la de proporcionar acceso a la salida de audio del dispositivo.

currentTime: Devuelve el tiempo en segundos transcurrido desde que el contexto es creado.

activeSourceCount: Devuelve el número del AudioBufferSourceNode que se está reproduciendo.

sampleRate: Devuelve la frecuencia (cuadros por segundo) con la que se procesa el audio.

listener: Devuelve un objeto AudioListener con propiedades y métodos para calcular la posición y orientación del oyente en una escena en 3D.

25.1.3 Fuentes de audio

Los nodos más importantes son probablemente los nodos de fuente de audio que representan el punto de partida de cada ruta en las figuras anteriores. Sin estas fuentes iniciales de audio, no habría ningún sonido que enviar a los altavoces. Estos tipos de nodos pueden ser creados a partir de diferentes fuentes: cachés de audio en la memoria, flujos medios o elementos multimedia. La API proporciona los métodos siguientes para obtener los objetos que representan a cada fuente:

createBufferSource(): Este método devuelve un objeto AudioBufferSourceNode que se crea a partir de un caché de audio en la memoria y ofrece sus propias propiedades y métodos para reproducir y configurar la fuente de audio.

createMediaStreamSource(stream): Este método devuelve un objeto MediaStreamAudioSourceNode, que se crea a partir de una secuencia de medios. El atributo stream es el stream o flujo generado por métodos como getUserMedia() (véase el Capítulo 9).

createMediaElementSource(elemento): Este método devuelve un objeto MediaElementAudioSourceNode a partir de un elemento multimedia. El atributo elemento es una referencia a un elemento <audio> o <video>.

Los métodos createMediaStreamSource() y createMediaElementSource() trabajan con los flujos de medios y elementos multimedia que tienen sus propios controles para reproducir, interrumpir o detener la reproducción, pero el método createBufferSource() funciona con sonidos almacenados en la memoria, y

necesita sus propios métodos y propiedades para reproducir y configurar la fuente. A continuación veremos de cuáles se trata.

start(tiempo, desplazamiento, duración): Este método comienza a reproducir el sonido asignado como fuente del nodo. El atributo tiempo determina cuándo se llevará a cabo la acción (en segundos). Los atributos desplazamiento y duración son opcionales. El primero determina qué parte del caché debe ser reproducida y el segundo, duración, determina los segundos.

stop(tiempo): Este método detiene la reproducción del sonido asignado como fuente del nodo. El atributo tiempo determina cuándo se llevará a cabo la acción (en segundos).

loop: Esta propiedad hace que el sonido asignado como fuente del nodo se reproduzca en bucle.

buffer: Esta propiedad declara como fuente del nodo a un caché de audio en la memoria.

El objeto devuelto por el método createBufferSource() no trabaja directamente con los archivos de audio. Los archivos tienen que ser descargados desde el servidor, almacenados en la memoria como cachés de audio y asignados al nodo de fuente de audio por la propiedad buffer. Para convertir los sonidos de un archivo de audio en un caché en la memoria, la API proporciona los métodos siguientes:

createBuffer(arraybuffer, mixToMono): Este método crea un caché de audio

a partir de datos binarios. El atributo arraybuffer consiste en los datos binarios del archivo de audio (tipo ArrayBuffer). El atributo mixToMono es un atributo booleano que determina si los canales de audio se mezclarán en un canal mono o no. El valor de este atributo es generalmente false.

decodeAudioData(arraybuffer, éxito, error): Este método crea de forma asíncrona un caché de audio a partir de datos binarios. El atributo arraybuffer consiste en los datos binarios del archivo de audio (tipo ArrayBuffer. El atributo éxito es la función que va a recibir y procesar el caché y el atributo error es una función que procesa los errores.

Las siguientes propiedades pueden recuperar información de las memorias intermedias de audio:

duration: Esta propiedad devuelve la duración de la memoria intermedia o caché en segundos.

length: Esta propiedad devuelve la longitud del caché en cuadros muestrales.

numberOfChannels: Esta propiedad devuelve el número de canales disponibles en el caché.

sampleRate: Esta propiedad devuelve la frecuencia de muestreo de la memoria intermedia en muestras por segundo.

25.1.4 Nodos de conexión

Hay una cosa más que tenemos que estudiar antes crear el primer ejemplo: cómo conectar los nodos. Los AudioNodes tienen conexiones de entrada y salida que nos permiten crear una ruta que el sonido debe seguir. La API proporciona dos métodos para organizar esta estructura:

connect(nodo): Este método conecta la salida de un nodo a la entrada de otro. La sintaxis es output.connect(input), donde output es una referencia al nodo de salida e input es una referencia al nodo que recibe el audio de esa salida.

disconnect(): Este método desconecta la salida del nodo. La sintaxis es output.disconnect(), donde output es una referencia para el nodo que será desconectado.

25.2 Sonidos para la Web

Llegó la hora de poner la teoría en práctica. Por lo general, los sonidos son parte de complejos códigos Javascript, similares a los necesarios para crear un videojuego en 3D o una aplicación de producción de audio, pero para los siguientes ejemplos usaremos un documento HTML modesto, que simplifique la creación y puesta en práctica de AudioNodes.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>API Web Audio</title>
  <script src="audio.js"></script>
</head>
<body>
  <section>
    <input type="button" id="boton" value="Reproducir"
 disabled="true">
  </section>
</body>
</html>
```

Código 25-1

Documento HTML para reproducir audio.

El documento HTML incluye el archivo audio.js con el código Javascript y un botón para reproducir los sonidos. Vamos a utilizar este documento para la mayoría de los ejemplos de este capítulo.

25.2.1 Dos nodos básicos

Como hemos explicado antes, una estructura básica de nodos exige un nodo para la fuente y otro para el destino. El nodo de destino es devuelto por la propiedad destino del contexto, pero la fuente de audio requiere un poco de trabajo. Tenemos que descargar el archivo de audio del servidor, convertir el contenido en un caché de audio, utilizar el caché como fuente del nodo y, finalmente, conectar ambos nodos para obtener sonido de los altavoces.

Para nuestro ejemplo vamos a descargar un archivo WAV utilizando Ajax y el objeto XMLHttpRequest.

```
var contexto;
function iniciar(){
  var buffer;
  var boton = document.getElementById('boton');
  boton.addEventListener('click', function(){ reproducir(buffer); });

  contexto = new webkitAudioContext();

  var url = 'gunshot.wav';
  var request = new XMLHttpRequest();
  request.responseType = "arraybuffer";
  request.addEventListener('load', function(){
    if(request.status == 200){
      buffer = contexto.createBuffer(request.response, false);
      boton.disabled = false;
    }
  });
  request.open("GET", url, true);
  request.send();
}
function reproducir(buffer){
  var nodoFuente = contexto.createBufferSource();
  nodoFuente.buffer = buffer;
  nodoFuente.connect(contexto.destination);
  nodoFuente.start(0);
}
addEventListener("load", iniciar);
```

Código 25-2

Reproducción de un caché de audio.

Como siempre, tenemos la función iniciar() para ejecutar la aplicación. La función comienza por obtener una referencia al botón HTML y añadir un detector para el evento click para ejecutar la función reproducir() cada vez que se hace clic en el botón. Inmediatamente después, se crea el contexto de audio mediante el constructor audioContext() y se almacena en la variable contexto. El archivo de audio gunshot.wav se carga junto con el Ajax.

El proceso de carga de este archivo es el mismo que aplicamos en los ejemplos del Capítulo 21, excepto que esta vez el tipo de respuesta está establecido como arraybuffer, que es el tipo de datos que necesitamos para crear el caché en la memoria.

Una vez que el archivo se ha descargado por completo y la propiedad estado de la solicitud devuelve el valor 200 (OK), se crea el caché a partir del valor de la propiedad response mediante el método createBuffer(). Ahora, el sonido del archivo es un caché de audio almacenado en la variable buffer y está listo para ser proporcionado como fuente para un nodo de fuente de audio.

Todo el sistema de audio, incluidos los nodos y las conexiones, tiene que ser construido cada vez que querramos reproducir un sonido. Así que la función iniciar() se encarga de descargar el archivo y convertir su contenido en un caché de audio en la memoria, pero el resto se hace mediante la función play().

Cada vez que se hace clic en el botón Reproducir, se crea el sistema de audio gracias a la función reproducir(). En primer lugar, se crea el nodo de fuente de audio mediante el método createBufferSource()y, a continuación, el caché de audio se asigna como valor de la propiedad buffer de este nodo. Este proceso enlaza el nodo al caché de audio que está en la memoria. Cuando este nodo esté listo será conectado al nodo de destino gracias al método connect(), y el sonido representado por el nodo nodo será finalmente reproducido con el método start().

Observe que el botón está desactivado inicialmente, pero se activa en la función iniciar(), una vez están listos los datos necesarios para crear el sistema de audio. Esto no es más que un proceso que decidimos seguir para simplificar el ejemplo. Una vez ha sido habilitado, cada vez que se hace clic en el botón, se crea de nuevo el sistema de audio y se reproduce el sonido del archivo de audio gunshot.wav. Puede seguir cualquier procedimiento para descargar los archivos e informar al resto del código cuando todo esté listo.

Por lo general, en aplicaciones profesionales, hay un código es responsable de descargar los archivos, establecer las variables e informar al código sobre la marcha. Más adelante vamos a ver algunos ejemplos de este modelo.

Hágalo usted mismo

Copie el documento del Código 25-1 en un archivo HTML y el Código 25-2 en un archivo Javascript llamado audio.js. Puede descargar el archivo gunshot.wav de nuestro sitio web (www.minkbooks.com/content/) o o utilizar su propio archivo de audio. Suba todos los archivos a su servidor y abra el documento en su navegador.

Importante

Para el momento de escribir esta manual, Google Chrome es el único navegador que tiene una implementación disponible de esta API. El navegador utiliza el el constructor webkitAudioContext() con prefijo para generar el contexto de audio. Vamos a aplicar este constructor en los ejemplos de este capítulo, pero usted debe comprobar el estado de la API para saber cuándo estará disponible en otros navegadores y el prefijo del constructor sigue siendo necesario. Para obtener más información, visite nuestra página web y siga los enlaces de este capítulo.

25.2.2 Bucles y tiempos

El método start() al final de la función reproducir() del último ejemplo fue llamada con un valor igual a 0. Cuando este valor se establece como 0 o es menor que el valor de la hora actual del contexto, el sonido empieza a sonar inmediatamente. Una vez reproducido, el método start() no puede volver a ser reproducido por el mismo nodo fuente de audio, se hace necesario crear un nuevo sistema de audio completo para hacerlo. Ésta es la razón por la que se construyó el sistema de audio en una sola función. A pesar de esta restricción, hay maneras diferentes de reproducir varias veces un sonido sin tener que llamar a la función reproducir() una y otra vez. Una alternativa es generar un bucle para este nodo fuente de audio estableciendo el valor de la propiedad bucle como true.

```
function reproducir(buffer){
  var nodoFuente = contexto.createBufferSource();
  nodoFuente.buffer = buffer;
  nodoFuente.loop = true;
  nodoFuente.connect(contexto.destination);
  nodoFuente.start(0);
  nodoFuente.stop(contexto.currentTime + 3);
}
addEventListener("load", iniciar);
```

Código 25-3

Reproducción en blucle del nodo fuente de audio.

El bucle se ejecutará indefinidamente hasta que sea llamado el método stop(). En el Código 25-3 se utiliza este método para detener el bucle después de tres segundos. La hora se ajusta con el valor devuelto por la propiedad currentTime. Al agregar el valor 3 a la hora actual, indicamos al navegador deje de reproducir el sonido al transcurrir tres segundos a partir de la hora actual.

Hágalo usted mismo

Hágalo usted mismo: Reemplace la función reproducir() del Código 25-2 con la nueva función presentada en el Código 25-3. Suba el archivo audio. js a su servidor y abra el documento del Código 25-1 en su navegador.

25.2.3 Crear AudioNodes

Ya hemos dicho que cada nodo del sistema de audio es un AudioNode. Aunque los nodos básicos que necesitamos para producir sonido son el nodo de fuente de audio y el nodo de destino, ambos creados en los ejemplos anteriores, estos no son los únicos nodos disponibles.

La API proporciona métodos para crear una variedad de AudioNodes para procesar, analizar e incluso crear sonidos:

createAnalyser(): Este método crea un nodo de análisis, es decir, del tipo AnalyserNode. Este tipo de nodos proporciona acceso a la información de la fuente de audio. El objeto devuelto incluye varias propiedades y métodos para este propósito. Frecuentemente es utilizado para visualizar el flujo de audio.

createGain(): Este método crea un nodo de volumen o GainNode. Este tipo de nodos establece la ganancia de la señal de audio (el volumen). El objeto devuelto proporciona la propiedad gain para establecer el volumen en valores comprendidos entre 0,0 y 1,0.

createDelay(maxDelayTime): Este método crea un nodo de retardo o, lo que es lo mismo, del tipo delayNode. Este tipo de nodo establece un retardo para la fuente de audio. El atributo opcional maxDelayTime declara el máximo de tiempo para el retardo en segundos. El objeto devuelto proporciona la propiedad delayTime para establecer el retardo en segundos.

createBiquadFilter(): Este método crea un nodo de filtro o del tipo BiquadFilterNode. Este tipo de nodo aplica filtros a la señal de audio. El objeto devuelto proporciona propiedades, métodos y también varias constantes para ajustar las características del filtro.

createWaveShaper(): Este método crea un nodo de distorsión o del tipo WaveShaperNode. Este tipo de nodo aplica un efecto de distorsión sobre la base de una curva de forma. El objeto devuelto proporciona la propiedad curve para declarar la curva de forma (tipo Float32Array).

createPanner(): Este método crea un nodo panorámico, o del tipo PannerNode. Estos nodos se utilizan para determinar la posición, orientación y velocidad del sonido en un espacio tridimensional. El efecto producido depende de los valores actuales establecidos para el oyente. El objeto devuelto proporciona varios métodos y propiedades para configurar los parámetros del nodo.

createConvolver(): Este método crea un nodo de convolución o, lo que es igual, del tipo ConvolverNode. Este tipo de nodo aplica un efecto de circunvolución a la señal de audio basado en una respuesta de impulso. El objeto devuelto proporciona dos propiedades: buffer y normalize. La propiedad buffer es necesaria para configurar el caché que se utilizará como respuesta de impulso, la propiedad normalize recibe un valor booleano para establecer si la respuesta al impulso se ampliará o no.

createChannelSplitter(numberOfOutputs): Este método crea un nodo de división de canales, es decir, del tipo ChannelSplitterNode. Este tipo de nodo genera salidas diferentes para cada canal de la corriente de audio. El atributo numberOfOutputs declara el número de resultados que serán generados.

createChannelMerger(numberOfInputs): Este método crea un nodo de combinación de canales, o del tipo ChannelMergerNode. Este tipo de nodos combina canales de múltiples flujos de audio en uno solo. El atributo numberOfInputs declara el número de entradas que se fusionan.

createDynamicsCompressor(): Este método crea un nodo de compresión o del tipo DynamicsCompressorNode. Este tipo de nodo aplica un efecto de compresión a la señal de audio. El objeto devuelto ofrece varias propiedades para configurar el efecto.

createOscillator(): Este método crea un nodo de oscilación o del tipo OscillatorNode. Este tipo de nodo genera formas de onda para la síntesis de audio. El objeto devuelto proporciona varias propiedades y métodos para configurar la forma de onda.

La aplicación de algunos de estos métodos requiere conocimientos en ingeniería de audio. Es un tema que va más allá del alcance de este libro, pero le mostraremos cómo implementar los métodos necesarios para la construcción de aplicaciones comunes de audio y de videojuegos en 2D y 3D.

25.2.4 AudioParam

Además de las propiedades y los métodos normales que cada objeto AudioNode proporciona para configurar el nodo, hay una interfaz definida por la API para establecer parámetros específicos.

La interfaz AudioParam es como una unidad de control vinculada al nodo, lista para ajustar sus valores en cualquier momento.

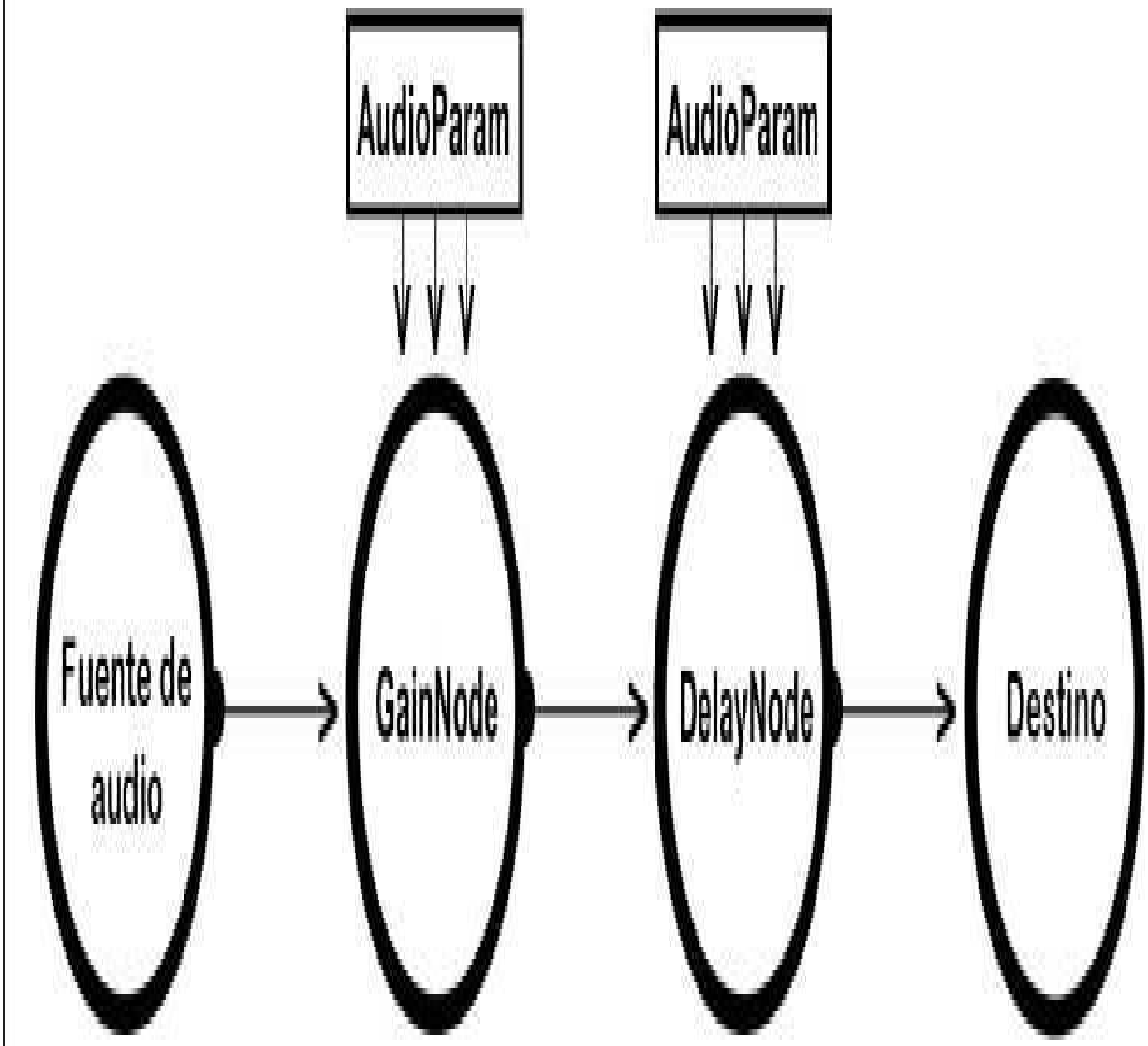

AudioParam
AudioParam
Fuente de audio
GainNode
DelayNode
Destino

Figura 25-3

Los nodos controlados por propiedades y métodos de AudioParam.

La interfaz puede establecer los valores inmediatamente o ser configurada para hacerlo más adelante siguiendo un horario específico. A continuación se enumeran las propiedades y métodos previstos para ello:

value: Esta propiedad declara el valor del parámetro al momento de definirlo.

setValueAtTime(valor, horaInicio): Este método establece el valor del parámetro en un momento dado. Los atributos valor y horaInicio declaran el nuevo valor y el tiempo en segundos, respectivamente.

linearRampToValueAtTime(valor, horaFinal): Este método cambia el valor del parámetro gradualmente del valor previo al especificado en los atributos. Los atributos valor y horaFinal declaran el nuevo valor y el tiempo final del proceso en segundos, respectivamente.

exponentialRampToValueAtTime(valor, horaFinal): Este método cambia el valor actual del parámetro de forma exponencial al valor especificado en los atributos. Los atributos valor y horaFinal declaran el nuevo valor y el tiempo final del proceso en segundos, respectivamente.

setTargetAtTime(objetivo, horaInicio, ConstanteTiempo): Este método cambia el valor del parámetro de forma exponencial al valor del atributo objetivo. El atributo horaInicio declara la hora de inicio del proceso, y timeConstant establece la velocidad a la que el valor anterior se aproxima al nuevo.

setValueCurveAtTime(valores, horaInicio, duración): Este método cambia el valor del parámetro por valores arbitrarios seleccionados de la matriz declarada en el atributo valores (del tipo Float32Array). Los atributos startTime y duración declaran la hora de inicio del proceso su la duración en segundos.

cancelScheduledValues(horaInicio): Este método cancela cualquier cambio anterior programado en el tiempo especificado por el atributo startTime.

25.2.5 GainNode

Generalmente, lo primero que queremos hacer cuando se reproduce un sonido es subir o bajar volumen. Para este propósito introduciremos un GainNode o nodo de ganancia entre el nodo de fuente de audio y el nodo de destino creado en los ejemplos anteriores.

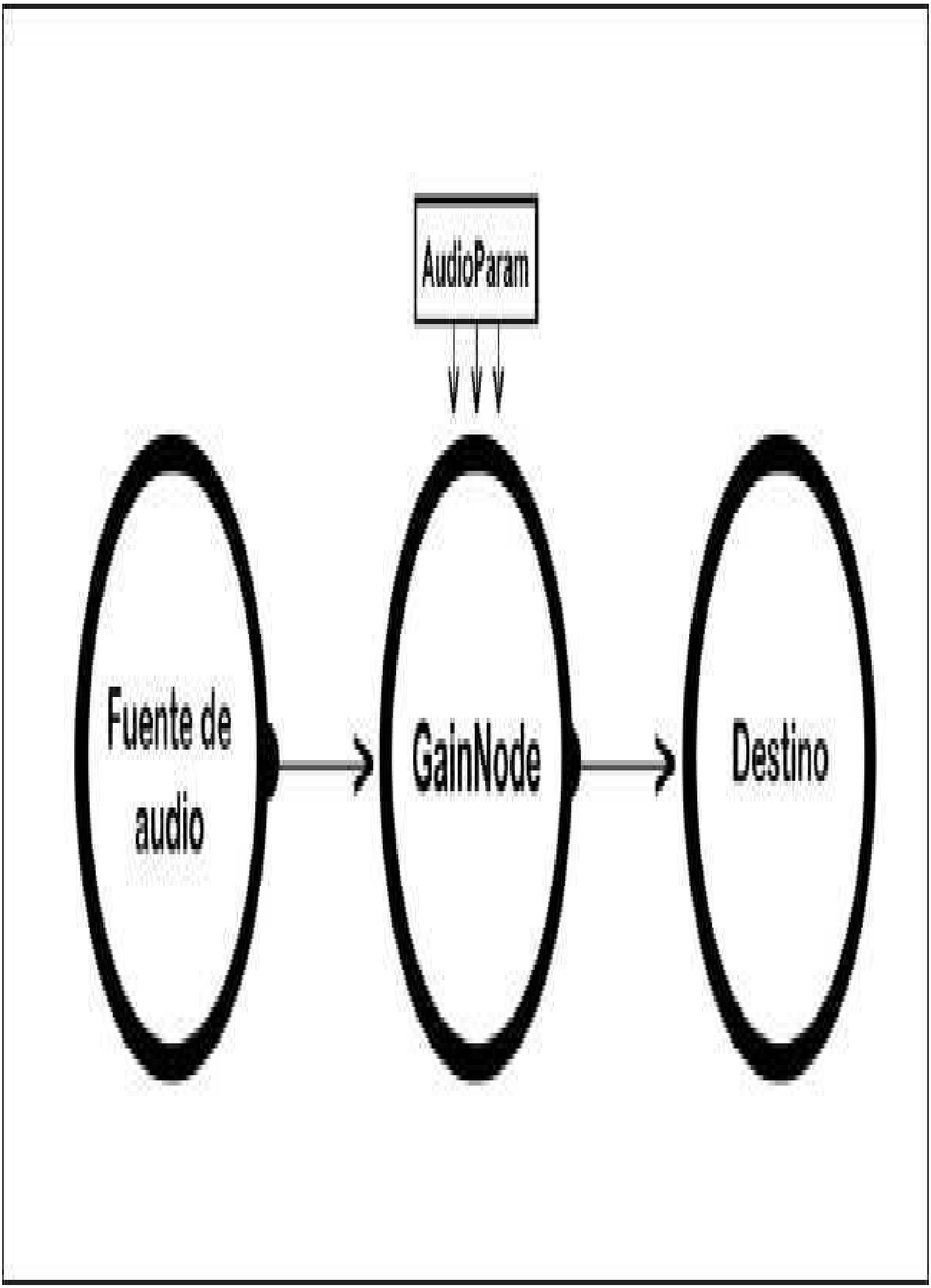

AudioParam
Fuente de audio
GainNode
Destino

Figura 25-4

Adición de un GainNode para controlar el volumen.

Cuando se añade un nuevo nodo a un sistema de audio, la estructura básica sigue siendo la misma. Simplemente se crea el nuevo nodo, se le proporcionan los valores de configuración y se conecta al resto de la estructura. En este caso, el nodo de fuente de audio puede ser conectado al GainNode y el GainNode al destino. Aquí está el código:

```
function reproducir(buffer){
  var nodoFuente = contexto.createBufferSource();
  nodoFuente.buffer = buffer;

  var nodoVolumen = contexto.createGain();
  nodoVolumen.gain.value = 0.2;

  nodoFuente.connect(nodoVolumen);
  nodoVolumen.connect(contexto.destination);
  nodoFuente.start(0);
}
addEventListener("load", iniciar);
```

Código 25-4

Bajar el volumen con un GainNode.

La nueva función reproducir() del Código 25-4 inserta un GainNode (en la variable nodoVolumen) en nuestro sistema de audio. El nodo es creado por el método createGain() y luego, utilizando la propiedad value de AudioParam, se le asigna un valor de 0,2 a la propiedad gain del nodo. Los valores posibles para esta propiedad van de 0,0 a 1. Por defecto, el valor de gain es 1, lo que quiere decir que en este ejemplo el volumen se reduce al 20%.

Hágalo usted mismo

Reemplace la función reproducir() del Código 25-2 con la nueva función presentada en el Código 25-4. Suba el archivo audio.js, el documento HTML del Código 25-1 y el archivo de audio a su servidor y abra el documento en su navegador.

Las conexiones realizadas al final del código incluyen el nuevo nodo del tipo GainNode. En primer lugar, el nodo fuente de audio es conectado a nodoVolumen, gracias a la línea (codigoFuente.connect(nodoVolumen)), y luego nodoVolumen es conectado al nodo de destino, gracias a la línea (nodoVolumen. connect(Contexto.destino)).

El valor de la propiedad gain en este ejemplo se establece como un valor fijo mediante la propiedad value, ya que el archivo de audio contiene solo el breve sonido de un disparo. Sin embargo, podríamos haber utilizado cualquiera de los métodos AudioParam para aumentar o reducir el volumen en diferentes momentos (por ejemplo, al final de una canción). Métodos tales como exponentialRampToValueAtTime() pueden seer utilizados en combinación con la propiedad duration de los cachés para mezclar canciones, atenuando una pista y la aumentando gradualmente el volumen de la siguiente.

25.2.6 DelayNode

El propósito de un nodo del tipo delayNode es retrasar el sonido durante el tiempo especificado por la propiedad DelayTime (y la interfaz AudioParam). En el siguiente ejemplo vamos a introducir este nodo para reproducir el sonido de la escopeta un segundo más tarde.

```
function play(mybuffer){
  var nodoFuente = contexto.createBufferSource();
  nodoFuente.buffer = buffer;

  nodoDelay = contexto.createDelay();
  nodoDelay.delayTime.value = 1;

  nodoFuente.connect(NodoDelay);
  nodoDelay.connect(contexto.destination);
  nodoFuente.start(0)
}
addEventListener("load", initiate);
```

Código 25-5

Uso de NodoDelay.

La función reproducir()del Código 25-5 reemplaza el GainNode introducido en el ejemplo anterior con un nodo de retraso o DelayNode, que es almacenado en la variable nodoDelay. Se establece un retraso de un segundo mediante la propiedad value de AudioParam, y las conexiones son establecidas siguiendo la misma trayectoria anterior: el nodo de fuente de audio está conectado a nodoDelay, y este nodo está conectado al nodo de destino.

Esto no es un gran efecto, pero es útil en algunas circunstancias. Un nodo de retraso produce mejores resultados cuando se combina con otros nodos, como sucede en el ejemplo siguiente:

```
function reproducir(buffer){
  var nodoFuente = contexto.createBufferSource();
  nodoFuente.buffer = buffer;

  nodoDelay = contexto.createDelay();
  nodoDelay.delayTime.value = 0.3;

  nodoVolumen = contexto.createGain();
  nodoVolumen.gain.value = 0.2;

  nodoFuente.connect(nodoDelay);
  nodoDelay.connect(nodoVolumen);
  nodoVolumen.connect(contexto.destination);
  nodoFuente.connect(contexto.destination);
  nodoFuente.start(0);
}
addEventListener("load", iniciar);
```

Código 25-6

Conseguir un efecto de eco con DelayNode.

En la función reproducir() del Código 25-6 se añade un nodo de volumen, es decir, del tipo GainNode, en el medio para reducir el volumen del sonido retardado y generar un efecto de eco. Se establecen dos rutas para la fuente de audio. En una ruta, el nodo de fuente de audio está conectado al nodo de retraso, el nodo de retraso al nodo de ganancia y el nodo de ganancia al nodo de destino. En esta ruta el sonido tendrá un volumen bajo y se reproducirá con un retraso de 0,3 segundos. La segunda ruta para la fuente es una conexión directa con el nodo de destino. El sonido en este camino se reproducirá inmediatamente y a todo volumen.

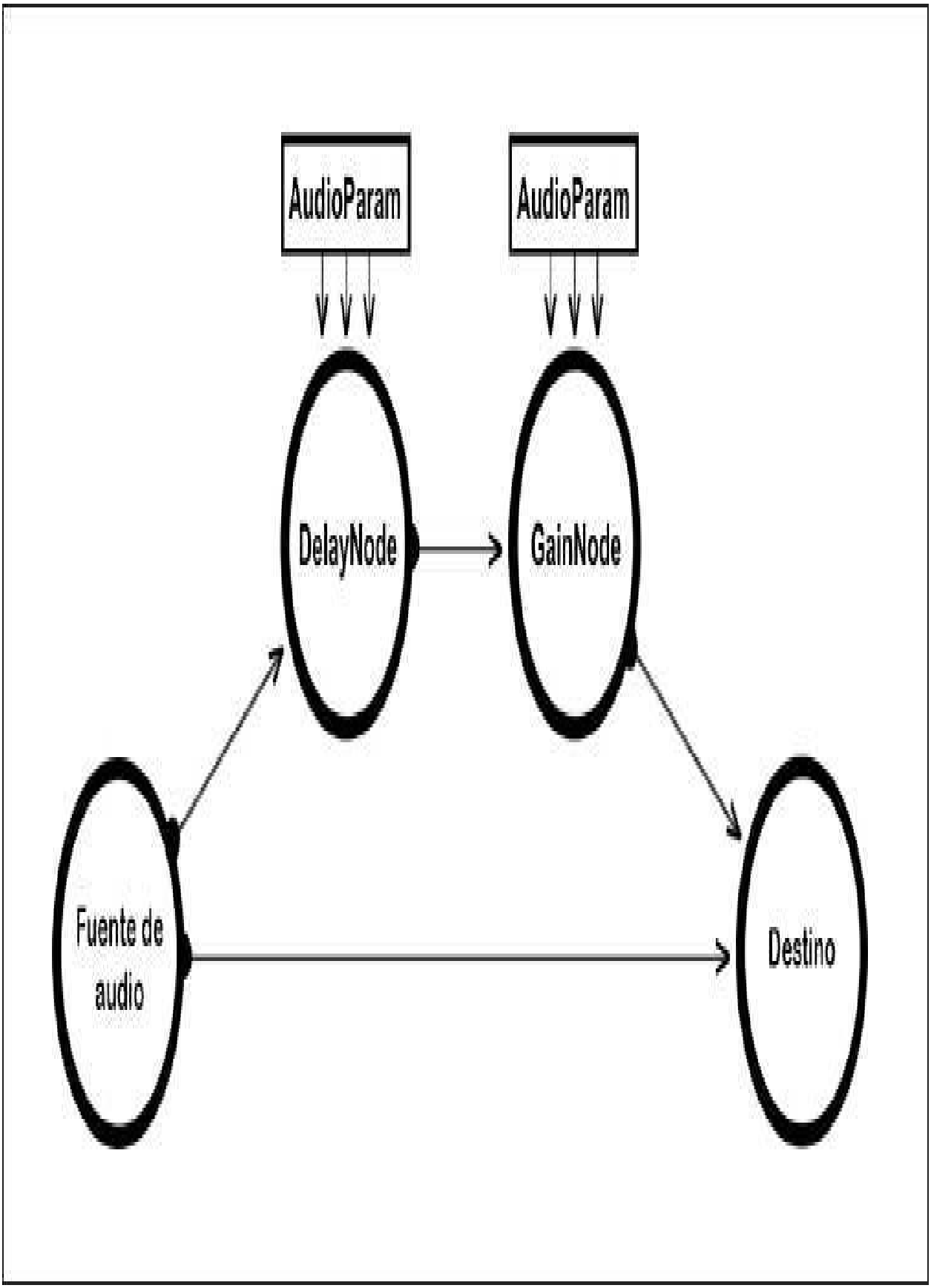

AudioParam
AudioParam
DelayNode
GainNode
Fuente de audio
Destino

Figura 25-5

Dos rutas para la misma fuente de audio.

Hágalo usted mismo

Reemplace una vez más la función reproducir() del Código 25-2 con las nuevas funciones presentadas en los códigos 25-5 y 25-6 para poner a prueba cada ejemplo. Suba el archivo audio.js, el documento HTML del Código 25-1 y el archivo de audio a su servidor y abra el documento en su navegador.

25.2.7 BiquadFilterNode

Un BiquadFilterNode o nodo de filtro biquad nos permite aplicar filtros comunes a la señal de audio. El nodo es creado por el método createBiquadFilter() y el tipo de filtro es seleccionado por la propiedad type. Los valores posibles para esta propiedad son LOWPASS (0), HIGHPASS (1), BANDPASS (2), LOWSHELF (3), HIGHSHELF (4), PEAKING (5), NOTCH (6) y ALLPASS (7). Para configurar el filtro, el nodo también incluye las propiedades frequency, Q y gain. El efecto sobre el filtro producido por los valores de estas propiedades depende del tipo de filtro.

```
function reproducir(buffer){
  var nodoFuente = contexto.createBufferSource();
  nodoFuente.buffer = buffer;


  var nodoFiltro = contexto.createBiquadFilter();
  nodoFiltro.type = 1;
  nodoFiltro.frequency.value = 1000;
  nodoFuente.connect(nodoFiltro);
  nodoFiltro.connect(contexto.destination);
  nodoFuente.start(0);
}
addEventListener("load", iniciar);
```

Código 25-7

Adición de un filtro con BiquadFilterNode.

En el ejemplo del Código 25-7 declaramos el tipo de filtro como LOWPASS (valor 1) y ajustamos la frecuencia de corte a 1000. Así se cortan algunas frecuencias del espectro y como resultado el sonido de bala parece el de un arma de juguete.

Hágalo usted mismo

Reemplace la función play() en el Código 25-2 con la nueva función presentada en el Código 25-7. Suba el archivo audio.js, el documento HTML del Código 25-1 y el archivo de audio a su servidor, y abra el documento en su navegador.

El valor de la propiedad de frecuencia se establece por la propiedad valor de AudioParam. Éste puede ser sustituido por cualquier método AudioParam para variar la frecuencia a lo largo del tiempo.

Importante

El efecto producido por los valores de las propiedades frequency, Q y gain

dependerá del filtro seleccionado. Algunos valores no producirán ningún efecto. Para obtener más información, lea la especificación de la API. Los enlaces están disponibles en www.minkbooks.com.

25.2.8 DynamicsCompressorNode

Los compresores suavizan el audio, reduciendo el volumen de los sonidos fuertes y elevando el volumen de los sonidos más bajos. Estos se conectan normalmente al final del sistema de audio con el fin de coordinar los niveles de audio de las fuentes para hacer que suenen unificados.

La API Web Audio incluye nodos de compresión o DynamicsCompressorNode para producir este efecto en la señal de audio. El nodo tiene varias propiedades para limitar y controlar la compresión: threshold (decibelios), knee (decibelios), ratio (valores del 1 al 20), reduction (decibelios), attack (segundos) y release (segundos).

```
function play(mybuffer){
  var nodoFuente = contexto.createBufferSource();
  nodoFuente.buffer = buffer;

  var nodoCompresor = contexto.createDynamicsCompressor();
  nodoCompresor.threshold = -60;
  nodoCompresor.ratio = 10;

  nodoFuente.connect(nodoCompresor);
  nodoCompresor.connect(contexto.destination);
  sourceNode.start(0);
}
addEventListener("load", initiate);
```

Código 25-8

Adición de un compresor dinámico.

El Código 25-8 muestra cómo crear y configurar el nodo de compresión dimámica, pero no es un ejemplo común de uso. Este tipo de nodo se implementa normalmente en el extremo de complicados sistemas de audio que incluyen varias fuentes de audio.

Hágalo usted mismo

Reemplace la función reproducir() del Código 25-2 con la nueva función del Código 25-8. Suba el archivo audio.js, el documento HTML del Código 25-1 y el archivo de audio a su servidor y abra el documento en su navegador.

25.2.9 ConvolverNode

Los nodos del tipo ConvolverNode aplican efectos de circunvolución a la señal de audio. Se utilizan normalmente para simular diferentes espacios acústicos y lograr distorsiones de sonido complejas, como una voz en un teléfono, por ejemplo. El efecto se consigue mediante el cálculo de una respuesta de impulsos usando un segundo archivo de audio. Este archivo de audio es por lo general una grabación realizada desde el espacio acústico real que estamos tratando de simular. Debido a este requisito, para poner en práctica el efecto, se tienen que descargar al menos dos archivos de audio, uno para la fuente y uno para la respuesta de impulso. Veamos un ejemplo:

```
var contexto;
var buffers = [];
function iniciar(){
  var boton = document.getElementById('boton');
  boton.addEventListener('click', function(){ reproducir(); });

  contexto = new webkitAudioContext();
  cargarbuffers('gunshot.wav', 0);
  cargarbuffers('garage.wav', 1);

  var control = function(){
    if(buffers.length >= 2){
      boton.disabled = false;
    }else{
      setTimeout(control, 200);
    }
  }
  control();
}
function cargarbuffers(url, id){
  var solicitud = new XMLHttpRequest();
  solicitud.responseType = "arraybuffer";
  solicitud.addEventListener('load', function(){
    if(solicitud.status == 200){
      buffers[id] = contexto.createBuffer(solicitud.response, false);
    }

  });
  solicitud.open("GET", url, true);
  solicitud.send();
}
function reproducir(){
  var nodoFuente = contexto.createBufferSource();
  nodoFuente.buffer = buffers[0];

  var nodoCircunvolucion = contexto.createConvolver();
  nodoCircunvolucion.buffer = buffers[1];

  nodoFuente.connect(nodoCircunvolucion);
  nodoCircunvolucion.connect(contexto.destination);
  nodoFuente.start(0);
}
addEventListener("load", iniciar);
```

Código 25-8

Adición de un compresor dinámico.

En el Código 25-9 recreamos todo el código Javascript para incluir un pequeño cargador que permita cargar los dos archivos de audio necesarios para el efecto (gunshot.wav y garage.wav). En esta ocasión tenemos que mover el código Ajax a una nueva función para descargar estos archivos de uno en uno y crear la función control() para controlar el proceso de descarga.

Hágalo usted mismo

Copie el Código 25-9 en el archivo audio.js, suba todos los archivos a su servidor y abra el documento del Código 25-1 en su navegador.

Al comienzo de la función iniciar(), la función cargarbuffers() es llamada dos veces para descargar cada uno de los archivos. Se crea un caché a partir de cada archivo y ambos son almacenados en la matriz buffers. Este proceso es verificado por la función control() y, cuando la longitud de la matriz es igual o mayor que 2, se habilita el botón Play.

La construcción del sistema de audio en la función reproducir() es similar al usado en ejemplos anteriores, excepto que esta vez son dos los nodos que requieren cachés de audio. Mediante la propiedad buffer de cada uno de ellos, les

asignamos el ítem correspondiente de la matriz buffers, y, finalmente, los conectamos.

25.1.10 PannerNode y sonido 3D

Las librerías de gráficos tridimensionales son hoy en día herramientas increíbles. La recreación de objetos reales en una pantalla de ordenador ha alcanzado un alto nivel de perfección y realismo y, como resultado de WebGL, esta experiencia increíble ya está disponible en la Web. Sin embargo, los gráficos no son suficientes para recrear el mundo real. Para esta tarea, el sonido también desempeña un papel significativo. En un mundo 3D, los sonidos no son estáticos: cambian cuando la posición de la fuente cambia y varían de acuerdo a la velocidad de la fuente y su orientación. Los nodos panorámicos o pannerNodes han sido diseñados específicamente para simular las variaciones en los sonidos producidos en una escena 3D.

Un nodo panorámico es similar a otros nodos, pero los sonidos deben ser configurados teniendo en cuenta los parámetros de un espacio 3D. El objeto que representa el nodo incluye las siguientes propiedades y métodos para este propósito:

panningModel: Esta propiedad indica el algoritmo a utilizar para colocar el audio de la escena 3D. Los valores posibles son EQUALPOWER, HRTF y SOUNDFIELD.

distanceModel: Esta propiedad especifica el algoritmo a utilizar para reducir el volumen del sonido de acuerdo con el movimiento de la fuente. Los valores posibles son LINEAR_DISTANCE, INVERSE_DISTANCE y EXPONENTIAL_DISTANCE.

refDistance: Esta propiedad especifica un valor de referencia para calcular la distancia entre la fuente y el oyente. Puede ser útil para adaptar el nodo a la escala de la escena 3D. El valor por defecto es 1.

maxDistance: Esta propiedad especifica la distancia máxima entre la fuente del sonido y el oyente. Después de este límite, el sonido mantendrá sus valores actuales.

rolloffFactor: Esta propiedad especifica la proporción en la que se reduce el volumen.

coneInnerAngle: Esta propiedad especifica el ángulo para fuentes de audio direccionales. Dentro de este ángulo, el volumen no se reduce.

coneOuterAngle: Esta propiedad especifica el ángulo para fuentes de audio direccionales. Fuera de este ángulo, el volumen se reduce a un valor determinado por la propiedad coneOuterGain.

setPosition(x, y, z): Este método establece la posición de la fuente de audio en relación con el oyente. Los atributos x,y y z declaran el valor de cada coordenada.

setOrientation(x, y, z): Este método establece la dirección de una fuente de audio direccional. Los atributos x, y y z declaran un vector de dirección.

setVelocity(x, y, z): Este método establece la velocidad y la dirección de la

fuente de audio. Los atributos x, y y z declaran un vector de dirección para representar la dirección y también la velocidad de la fuente.

Como habrá deducido por las descripciones de algunos de estos métodos y propiedades, la mayoría de los sonidos de una escena 3D son definidos en relación a la posición del oyente. Debido a que solo hay un oyente, sus características son definidas para el contexto de audio y no para un nodo en particular. El objeto que representa al oyente puede ser accedido a través de la propiedad listener y proporciona las siguientes propiedades y métodos para su configuración:

dopplerFactor: Esta propiedad declara un valor para configurar el cambio de tono para el efecto Doppler.

speedOfSound: Esta propiedad especifica la velocidad del sonido en metros por segundo. Se utiliza para calcular el desplazamiento Doppler. El valor por defecto es 343.3.

setPosition(x, y, z): Este método establece la posición del oyente. Los atributos x, y y z declaran el valor de cada coordenada.

setOrientatio(x, y, z, xUp, yUp, zUp): Este método establece la dirección en la que está orientado el oyente. Los atributos x, y y z declaran un vector de dirección frente al oyente, y los atributos xup, yup y zup declaran un vector de dirección sobre del oyente.

setVelocity(x, y, z): Este método establece la velocidad y dirección del oyente. Los atributos x, y y z declaran un vector de dirección para

representar la dirección y también la velocidad del oyente.

Vamos a ver cómo utilizar toda esta teoría en una aplicación 3D usando la biblioteca Three.js. El documento HTML que hemos estado utilizando hasta ahora en este capítulo tiene que ser modificado para incluir el archivo three.min.js y el elemento <canvas>.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>API Web Audio</title>
  <script src="three.min.js"></script>
  <script src="audio.js"></script>
</head>
<body>
  <section>
   <canvas id="lienzo" width="500" height="400"></canvas>
  </section>
</body>
</html>
```

Código 25-10

Documento HTML que incluye la biblioteca Three.js

El código de este ejemplo dibujará un cubo en la pantalla, que puede ser movido hacia adelante y hacia atrás con la rueda del ratón. A continuación presentamos el código Javascript que deberá contener el archivo audio.js:

```
var contexto, nodoPanoramico, renderer, escena, camara, malla;
  function iniciar(){
  var lienzo = document.getElementById('lienzo');
  var anchura = lienzo.width;
  var altura = lienzo.height;

  renderer = new THREE.WebGLRenderer({canvas: lienzo});
  escena = new THREE.Scene();
  camara = new THREE.PerspectiveCamera(45, anchura/altura, 0.1,
 10000);
  camara.position.set(0, 0, 150);

  var geometria = new THREE.CubeGeometry(50, 50, 50);
  var material = new THREE.MeshPhongMaterial({color: 0xCCCCFF});
  malla = new THREE.Mesh(geometria, material);
  malla.rotation.y = 0.5;
  malla.rotation.x = 0.5;
  escena.add(malla);

  var luz = new THREE.SpotLight(0xFFFFFF, 1);
  luz.position.set(0, 100, 250);
  escena.add(luz);

  contexto = new webkitAudioContext();
  contexto.listener.setPosition(0, 0, 150);

  var url = 'engine.wav';
  var solicitud = new XMLHttpRequest();
  solicitud.responseType = "arraybuffer";
  solicitud.addEventListener('load', function(){
   if(solicitud.status == 200){
     var buffer = contexto.createBuffer(solicitud.response, false);
     reproducir(buffer);
     lienzo.addEventListener('mousewheel', mover, false);
     renderer.render(escena, camara);
   }
  });
  solicitud.open("GET", url, true);
  solicitud.send();
}
function reproducir(buffer){
  var nodoFuente = contexto.createBufferSource();
  nodoFuente.buffer = buffer;
  nodoFuente.loop = true;

  nodoPanoramico = contexto.createPanner();
  nodoPanoramico.refDistance = 100;
  nodoFuente.connect(nodoPanoramico);
  nodoPanoramico.connect(contexto.destination);
  nodoFuente.start(0);
}
function mover(e){
  malla.position.z += e.wheelDeltaY / 5;
  nodoPanoramico.setPosition(malla.position.x, malla.position.y,
 malla.position.z);
  renderer.render(escena, camara);
}
addEventListener('load', iniciar, false);
```

Código 25-11

Cálculo de la posición del sonido en una escena 3D.

La función iniciar() del Código 25-11 inicializa todos los elementos de la escena 3D y descarga el archivo engine.wav que contiene el sonido de nuestro cubo. Tenga en cuenta que después de crear el contexto de audio con el constructor audioContext(), la posición del oyente será establecida con el método setPosition(). Esto lo hemos hecho en el proceso de inicialización debido a que en este ejemplo el oyente mantiene la misma posición todo el tiempo, pero esto probablemente no sea así en otras aplicaciones.

Después de que el archivo es descargado y es creado el caché de audio, se añade un detector para el evento mousewheel que dispara la función mover() cada vez que se hace girar la rueda del ratón. Esta función actualiza la posición del cubo en el eje z según la dirección en que se mueve la rueda y establece las nuevas coordenadas para la fuente de audio. El efecto hará que parezca que el cubo realmente es la fuente del sonido.

En la función reproducir() se crea el nodo de paneo o PannerNode y se inserta en el sistema de audio. Usamos la propiedad refDistance para establecer los valores para el nodo en relación a la escala de la escena 3D.

Hágalo usted mismo

Cree un nuevo archivo HTML con el Código 25-10, copie el Código 25-11 en el archivo audio.js, suba todos los archivos a su servidor y abra el documento en su navegador. Los enlaces para descargar el archivo engine. wav están disponibles en www.minkbooks.com/content/.

25.1.11 AnalyserNode

Sería extremadamente difícil aplicar todas las herramientas proporcionadas por la API Web Audio en un entorno profesional si no fuéramos capaces de visualizar los resultados en pantalla. Para ello, la API incluye un nodo de análisis o AnalyserNode. Este nodo implementa un algoritmo llamado FFT (Fast Fourier Transform) que convierte la forma de onda de la señal de audio en una matriz de valores que representan la magnitud y la frecuencia en un período de tiempo específico. Los valores devueltos pueden ser utilizados para analizar la señal o crear gráficos que muestren sus valores en la pantalla. Un nodo de análisis proporciona propiedades y métodos para recuperar y procesar la información:

fftsize: Esta propiedad especifica el tamaño de la FFT (el tamaño del bloque de datos a ser analizado). El valor debe ser una potencia de 2 (por ejemplo, 128, 256, 512, 1024, etc.).

frequencyBinCount: Esta propiedad devuelve el número de valores de frecuencia proporcionados por la FFT.

minDecibels: Esta propiedad especifica el valor de potencia mínima de la FFT.

maxDecibels: Esta propiedad especifica el valor de potencia máxima de la FFT.

smoothingTimeConstant: Esta propiedad declara un período de tiempo en

el que el analizador obtendrá un valor medio de las frecuencias. Toma un valor de 0 a 1.

getFloatFrequencyData(matriz): Este método recupera de la señal de audio los datos de frecuencia actual y los almacena en el atributo matriz como valores de punto flotante. El atributo matriz es una referencia a una matriz de punto flotante ya creada.

getByteFrequencyData(matriz): Este método recupera los datos de la frecuencia actual de la señal de audio y los almacena en el atributo matriz como valores de bytes sin signo. El atributo matriz es una referencia a una matriz de bytes sin signo ya creada.

getByteTimeDomainData(matriz): Este método recupera los datos de forma de onda actual de la señal de audio y los almacena en el atributo matriz como valores de punto flotante. El atributo matriz es una referencia a una matriz de punto flotante ya creada.

Los datos producidos por estos métodos y propiedades pueden ser utilizados para actualizar un gráfico en la pantalla y muestran la evolución de la señal de audio. Para ver cómo implementar un nodo de análisis, vamos a utilizar un elemento <canvas> para visualizar el sonido tomado de un elemento <video>. Aquí está el documento HTML de este ejemplo:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>API Web Audio</title>
  <style>
   section{
    float: left;
   }
  </style>
  <script src="audio.js"></script>
</head>
<body>
  <section>
   <video id="medios" width="483" height="272" controls>
    <source src="http://www.minkbooks.com/content/trailer2.mp4">
    <source src="http://www.minkbooks.com/content/trailer2.ogg">
   </video>
  </section>
  <section>
   <canvas id="lienzo" width="500" height="272"></canvas>
  </section>
</body>
</html>
```

Código 25-12

Documento HTML para probar AnalyserNode.

Como ya hemos explicado antes, cuando la fuente es un elemento multimedia, el nodo de fuente de audio debe ser creado con el método createMediaElementSource(). En el siguiente código de Javascript se aplicarán éste y el método createAnalyser()para obtener los nodos que necesitamos para nuestro sistema de audio:

```
var lienzo, contexto, nodoAnalisis;
function iniciar(){
  var video = document.getElementById('medios');
  var elem = document.getElementById('lienzo');

  lienzo = elem.getContext('2d');
  contexto = new webkitAudioContext();

  var nodoFuente = contexto.createMediaElementSource(video);
  nodoAnalisis = contexto.createAnalyser();
  nodoAnalisis.fftSize = 512;
  nodoAnalisis.smoothingTimeConstant = 0.9;

  nodoFuente.connect(nodoAnalisis);
  nodoAnalisis.connect(contexto.destination);
  mostrargrafico();
}
function mostrargrafico(){
  var data = new Uint8Array(nodoAnalisis.frequencyBinCount);
  nodoAnalisis.getByteFrequencyData(data);

  lienzo.clearRect(0, 0, 500, 400);
  lienzo.beginPath();
  for(var f = 0; f < nodoAnalisis.frequencyBinCount; f++){
   lienzo.fillRect(f * 5, 272 - data[f], 3, data[f]);
  }
  lienzo.stroke();

  webkitRequestAnimationFrame(mostrargrafico);
}
addEventListener('load', iniciar, false);
```

Código 25-13

Dibujar el gráfico de un elemento <canvas>.

Para limitar el tamaño de los datos recuperados y suavizar el gráfico en la pantalla, en la secuencia de comandos de Código 25-13, el tamaño de la FFT se fija en 512 y se establece un período de tiempo de 0,9 con la propiedad smoothingTimeConstant. Después de que los nodos son creados, configurados y conectados, el sistema de audio está listo para proporcionar información sobre la señal de audio. La función mostrargrafico() es creada para procesar los datos. Esta función crea una matriz vacía del tipo Uint8Array con el tamaño determinado por la propiedad frequencyBinCount y llama al método getByteFrequencyData() para rellenar la matriz con los valores de la señal de audio. Debido al tipo de matriz, los valores almacenados serán números enteros desde 0 a 256 (enteros de 8 bits). En el bucle for, a continuación, se utilizan estos valores para calcular el tamaño de las barras que representan las frecuencias correspondientes y dibujar en el lienzo.

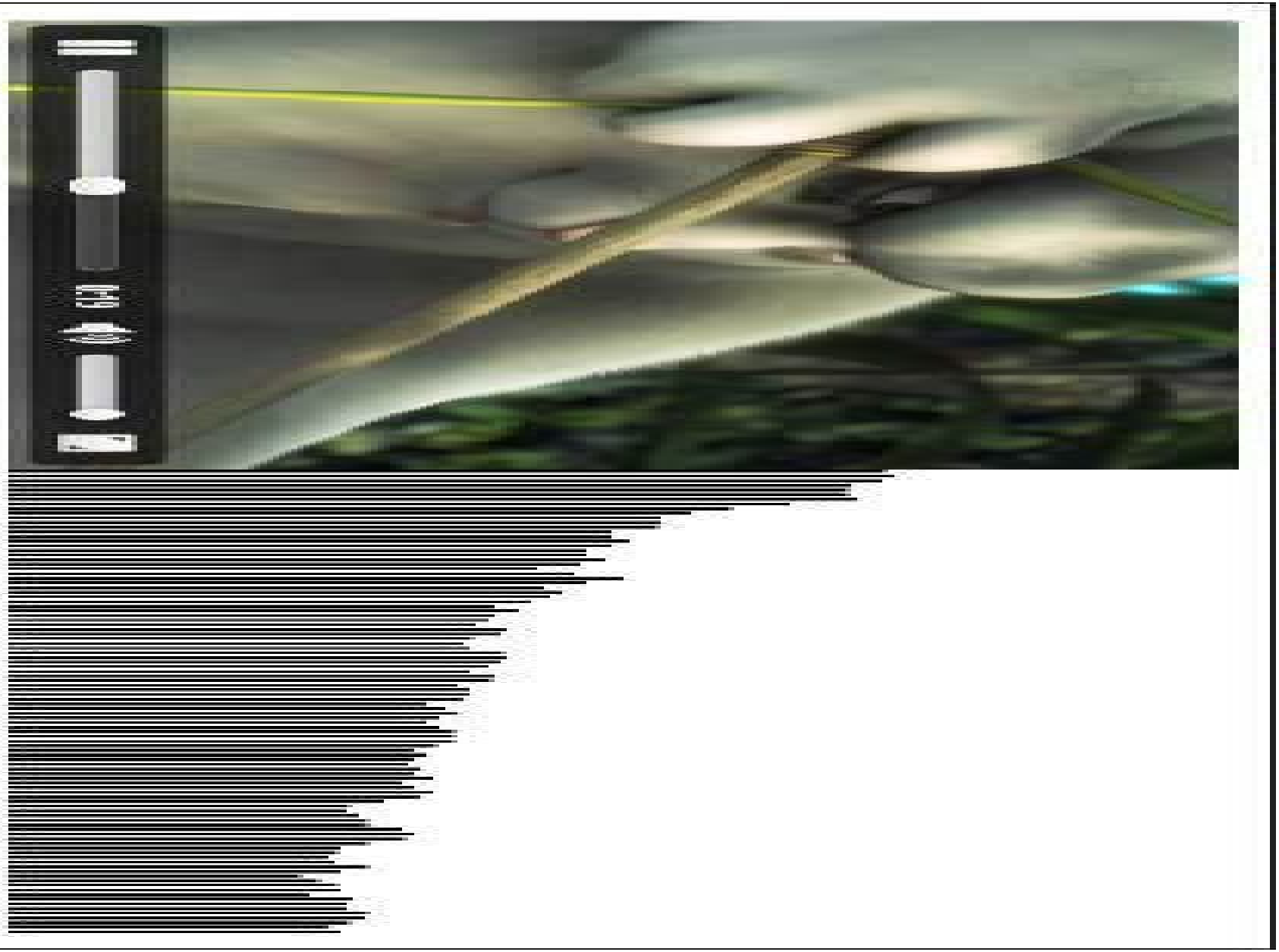


Figura 25-6

Gráfico de audio para el video creado a partir de AnalyserNode. © 2008, Blender Foundation www.bigbuckbunny.org.

Conceptos básicos

Las variables de Javascript generalmente son definidas sin tener en cuenta el tipo de datos con el que se va a trabajar, pero algunos nuevos métodos solo toman los valores proporcionados si son de un tipo específico. EL constructor Uint8Array(), implementado en el ejemplo de Código 25-13, es solo uno de los constructores introducidos por el lenguaje para proporcionar nuevos tipos de matrices para satisfacer los requisitos de las API de HTML5. Para obtener más información sobre el tema, visite nuestro sitio web y siga los enlaces de este capítulo.

Hágalo usted mismo

Cree un nuevo archivo HTML con el Código 25-12, copie el Código 25-13 en el archivo audio.js y ábralo en su navegador.

26 API Web Workers

26.1 Hacer el trabajo duro

Javascript se ha convertido en la herramienta principal para la creación de aplicaciones exitosas en la Web. Como explicamos en el Capítulo 4, ya no es solo una alternativa para la creación de buenos (y a veces molestos) trucos para páginas web. El lenguaje es ya una parte esencial de la Web y una tecnología que todos deben entender y aplicar.

Javascript ha alcanzado el estatus de un lenguaje de uso general y, como resultado, se ha visto obligado a ofrecer características elementales que por naturaleza no tenía. Desde su creación, el lenguaje fue diseñado para que fuera procesado un código a la vez. La incapacidad de procesar simultáneamente múltiples códigos (multithreading) reduce la eficacia y limita el alcance de la tecnología, haciendo de esta forma que la emulación de aplicaciones de escritorio en la Web sea una tarea imposible.

La API Web Workers ha sido diseñada con el propósito específico de convertir al lenguaje Javascript en multithreading y, así, solucionar el problema. Ahora, con HTML5, es posible ejecutar en segundo plano trabajos cuya ejecución toma mucho tiempo, mientras se ejecuta el código principal de forma ininterrumpida en la página web, sin dejar de procesar las entradas del usuario o perder la capacidad de respuesta del documento.

26.1.1 Crear un trabajador

El funcionamiento de la API Web Worker es simple: se construye un worker (trabajador) en un archivo separado de Javascript, y los códigos se comunican entre sí a través de mensajes. Pero antes de comunicarnos con el trabajador, tenemos que obtener un objeto que apunte al archivo en el que se encuentra el código del trabajador.

Worker(scriptURL): Este constructor devuelve un objeto Worker. El atributo scriptURL es la dirección URL del archivo que contiene el código que será procesado en segundo plano (worker).

Por lo general, el mensaje enviado al trabajador a partir del código principal es la información que se desea procesar, y los mensajes enviados de vuelta por el trabajador representan el resultado de este procesamiento. Para enviar y recibir estos mensajes, la API saca provecho de las técnicas implementadas al estudiar la API Web Messaging. Los eventos y métodos que ya conocemos son utilizados para enviar y recibir mensajes de un código a otro:

postMessage(mensaje): Este método ya fue estudiado en el Capítulo 22, dedicado a la la API Web Messaging, pero ahora se implementa para el objeto Worker. Envía un mensaje cuyo emisor o receptor es el código trabajador. El atributo mensaje es cualquier valor Javascript, como una cadena o un número, o también datos binarios, como un objeto File o un ArrayBuffer que representa el mensaje que se va a transmitir.

message: Este evento detecta los mensajes enviados al código. Al igual que

ocurría con el método postMessage(), puede ser aplicado al trabajador o al código principal. Devuelve un objeto con la propiedad data para recuperar el contenido del mensaje.

26.1.2 Enviar y recibir mensajes

Para aprender se comunican los códigos trabajadores y el código principal entre ellos, vamos a usar un sencillo documento HTML que enviará nuestro nombre como un mensaje para el código trabajador e imprimir la respuesta. Incluso un ejemplo de Web Workers tan básico como éste requiere de al menos tres archivos: el documento principal, el código principal y el archivo con el código Javascript para el trabajador.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Web Workers</title>
  <link rel="stylesheet" href="webworkers.css">
  <script src="webworkers.js"></script>
</head>
<body>
  <section id="cajaformulario">
    <form name="formulario">
      <label for="nombre">Nombre: </label><br>
      <input type="text" nombre="nombre" id="nombre"><br>
      <input type="button" id="boton" value="Enviar">
    </form>
  </section>
  <section id="cajadatos"></section>
</body>
</html>
```

Código 26-1

Documento HTML para probar la API Web Workers.

En el documento principal del Código 26-1, se incluye un archivo CSS llamado webworkers.css con las siguientes reglas:

```
#cajaformulario{
   float: left;
   padding: 20px;
   border: 1px solid #999999;
}

#cajadatos{
   float: left;
   width: 500px;
   margin-left: 20px;
   padding: 20px;
   border: 1px solid #999999;
}
```

Código 26-2

El estilo de las cajas (webworkers.css).

El Javascript para el documento principal tiene que ser capaz de enviar la información que se desea para ser procesada por el trabajador. Este código también tiene que ser capaz de escuchar la respuesta.

```
var worker, cajadatos;
function iniciar(){
  cajadatos = document.getElementById('cajadatos');
  var boton = document.getElementById('boton');
  boton.addEventListener('click', enviar);

  worker = new Worker('worker.js');
  worker.addEventListener('message', recibir);
}
function enviar(){
  var nombre = document.getElementById('nombre').value;
  worker.postMessage(nombre);
}
function recibir(e){
  cajadatos.innerHTML = e.data;
}
addEventListener('load', iniciar);
```

Código 26-3

Cargar el worker (webworkers.js).

El Código 26-3 será el contenido del archivo webworkers.js. Después de crear en la función iniciar() las referencias necesarias a cajadatos y al botón Enviar, se construye el objeto Worker. El constructor Worker() toma el archivo worker.js como el archivo trabajador y devuelve un objeto Worker con esta referencia. Cada interacción con el objeto, en realidad, ser una interacción con el código de este archivo particular.

Una vez que tenemos el objeto apropiado, se añade un detector para el evento message que detecta los mensajes que vienen del archivo trabajador. Cuando se recibe un mensaje, se llama a la función recibir() y el valor de la propiedad data (el mensaje) es mostrado en la pantalla.

La otra parte de la comunicación se lleva a cabo gracias a la función enviar(). Cuando el usuario hace clic en el botón Enviar, se envía el valor de la entrada nombre como un mensaje para el trabajador mediante el método postMessage().

Con las funciones recibir() y enviar() a cargo de las comunicaciones, estamos listos para enviar mensajes al trabajador y procesar sus respuestas. Ahora es el momento de preparar el trabajador:

```
addEventListener('message', recibir);

function recibir(e){
  var respuesta = 'Su nombre es '+e.data;
  postMessage(respuesta);
}
```

Código 26-4

Crear del trabajador (worker.js).

Igual que el Código 26-3, el código del trabajador tiene que detectar constantemente los mensajes que proceden del código principal usando el evento message. La primera línea del Código 26-4 añade al trabajador un detector de este evento. Cada vez que se dispara el evento (es decir, que se recibe un mensaje), se ejecuta la función recibir(). En esta función, se añade el valor de la propiedad data a un texto predefinido y se envía de nuevo al código principal utilizando de nuevo el método postMessage().

Hágalo usted mismo

Compare el Código 26-3 (el código principal) con el Código 26-4 (código trabajador). Vea cómo funciona el proceso de comunicación y cómo se aplican el mismo método y el mismo evento en ambos códigos para este fin. Utilice los códigos 26-1, 26-2, 26-3 y 26-4 para crear los archivos correspondientes a cada uno, súbalos a su servidor y abra el documento HTML en su navegador.

Este trabajador es, por supuesto, elemental. Realmente no procesa nada. Simplemente construye una cadena a partir del mensaje recibido e inmediatamente la envía de vuelta como respuesta. Sin embargo, este ejemplo es útil para ayudarle a comprender cómo se comunican entre sí los códigos y cómo puede usted tomar ventaja de esta API.

A pesar de su sencillez, hay algunas cosas importantes que debe considerar antes de crear sus propios trabajadores. Los trabajadores solo pueden comunicarse a través de mensajes. Además, los códigos trabajadores no pueden acceder al documento o manipular ningún elemento HTML, y las funciones Javascript o variables del código principal no son accesibles a estos. Los códigos trabajadores son como los códigos enlatados, autorizados únicamente a procesar la información recibida a través de mensajes y enviar el resultado de vuelta utilizando el mismo mecanismo.

26.1.3 Detectar errores

A pesar de todas las limitaciones mencionadas, los trabajadores son flexibles y potentes. Es posible utilizar las funciones, los métodos nativos y las API desde el interior de un código trabajador.

Teniendo en cuenta lo complejo que puede llegar a ser un trabajador, la API Web Workers incorpora un evento específico para comprobar si hay errores y devolver toda la información posible sobre la situación.

error: Este evento es disparado por el objeto Worker en el código principal cada vez que se produce un error en el código trabajador. Utiliza tres propiedades para proporcionar información: message.filename y lineno. La propiedad message representa el mensaje de error. Es una cadena que indica qué salió mal. La propiedad filename muestra el nombre del archivo que contiene el código que provocó el error. Es útil cuando el código worker carga archivos externos, como veremos más adelante. Finalmente, la propiedad lineno devuelve el número de línea en el que se produjo el error.

Vamos a crear un código que muestre los errores devueltos por el trabajador:

```
var worker, cajadatos;
function iniciar(){
  cajadatos = document.getElementById('cajadatos');
  var boton = document.getElementById('boton');
  boton.addEventListener('click', enviar);

  worker = new Worker('worker.js');
  worker.addEventListener('error', error);
}

function enviar(){
  var nombre = document.getElementById('nombre').value;
  worker.postMessage(nombre);
}

function error(e){
  cajadatos.innerHTML = 'Error: ' + e.message + '<br>';
  cajadatos.innerHTML += 'Nombre del archivo: ' + e.filename +
 '<br>';
  cajadatos.innerHTML += 'Número de línea: ' + e.lineno;
}
addEventListener('load', iniciar);
```

Código 26-5

Detectar el evento error (webworkers.js).

Las declaraciones Javascript del trabajador (Código 26-5) son similares a las del código principal (Código 26-3). Se construye un trabajador que solo utiliza el evento error porque en esta ocasión no queremos escuchar las respuestas del trabajador sino comprobar si hay errores. Es inútil, por supuesto, pero le mostrará cómo se devuelven los errores y qué tipo de información se ofrece en estas situaciones.

Para generar deliberadamente un error, vamos a llamar a una función que no existe desde el trabajador:

```
addEventListener('message', recibir);
```

```
function recibir(e){
    prueba();
}
```

Código 26-6:

Producir un error (worker.js)

En el código trabajador, tenemos que utilizar el evento message para detectar los mensajes que vienen del código principal, que es el que comienza el proceso. Cuando se recibe un mensaje, se ejecuta la función recibir()y ésta llama a la inexistente función prueba(), lo que genera un error.

Tan pronto como se produce el error, se dispara el evento en el código principal, y se ejecuta la función error() de éste, que muestra en la pantalla los valores de las tres propiedades proporcionadas por el evento. Compruebe en el Código 26-5 cómo la función toma y utiliza esta información.

Hágalo usted mismo

Para este ejemplo estamos trabajando con el documento HTML y las reglas CSS de los códigos 26-1 y 26-2. Copie el Código 26-5 en el archivo webworkers.js y el Código 26-6 en el archivo worker.js. Abra el documento del Código 26-1 en su navegador y envíe cualquier cadena aleatoria al archivo trabajador usando el formulario para activar el proceso. El error devuelto por el trabajador se mostrará en la pantalla.

26.1.4 Detener el trabajador

Los trabajadores son unidades especiales de código que siempre se ejecutan en segundo plano, esperando a que la información sea procesada. La mayoría de las veces, son necesarios solo en determinadas circunstancias y por períodos limitados de tiempo. Como por lo general sus servicios no son requeridos todo el tiempo, es una buena práctica para detener o dar por terminado su procesamiento cuando ya no los necesitamos más. La API proporciona dos métodos diferentes para este propósito:

terminate(): Este método detiene al trabajador desde el código principal.

close(): Este método detiene al trabajador desde su propio código.

Cuando un trabajador se detiene, se interrumpe cualquier proceso que esté ejecutando, y se descarta cualquier tarea del bucle de eventos. Para probar ambos métodos, vamos a crear una pequeña aplicación que funciona exactamente como nuestro primer ejemplo, pero que también responde a dos comandos específicos: cerrar1 y cerrar2. Si las cadenas cerrar1 o cerrar2 son enviadas desde el formulario, el código trabajador será detenido desde el código principal o el código trabajador usando los métodos terminate() o close(), respectivamente.

```
var worker, cajadatos;
function iniciar(){
  cajadatos = document.getElementById('cajadatos');
  var boton = document.getElementById('boton');
  boton.addEventListener('click', enviar);

  worker = new Worker('worker.js');
  worker.addEventListener('message', recibir);
}
function enviar(){
  var nombre = document.getElementById('nombre').value;
  if(nombre == 'cerrar1'){
    worker.terminate();
    cajadatos.innerHTML = 'Trabajador detenido';
  }else{
    worker.postMessage(nombre);
  }
}
function recibir(e){
  cajadatos.innerHTML = e.data;
}
addEventListener('load', iniciar);
```

Código 26-7

Terminar el trabajador a partir del código principal (webworkers.js)

La única diferencia entre la secuencia de comandos de Código 26-7 y la del Código 26-3 es la adición de la declaración if para verificar la inserción del comando cerrar1. Si se inserta este comando en el formulario, se ejecuta el método terminate() y se muestra un mensaje en la pantalla que indica la que el trabajador se ha detenido. Por otro lado, si la cadena es diferente del comando esperado, se envía como un mensaje para el trabajador.

El código trabajador llevará a cabo una tarea similar. Si el mensaje recibido contiene la cadena cerrar2, el trabajador detendrá su propia ejecución mediante el método close() y si no es así, enviará un mensaje en respuesta.

```
addEventListener('message', recibir);


function recibir(e){
  if(e.data == 'cerrar2'){
    postMessage('Trabajador terminado');
    close();
  }else{

    var answer = 'Su nombre es: ' + e.data;
    postMessage(answer);
  }
}
```

Código 26-8

Detener el trabajador desde su propio código.

Hágalo usted mismo

Utilice el mismo documento HTML y las reglas CSS presentados en los Códigos 26-1 y 26-2. Copie el Código 26-7 en el archivo webworkers.js y el Código 26-8 en el archivo worker.js. Abra el documento en el navegador y, utilizando el formulario, envíe los comandos cerrar1 o cerrar2. Después de introducir cualquiera de los dos comandos, el trabajador no responderá más.

26.1.5 API asíncronas

Los códigos trabajadores pueden tener limitaciones en cuanto al trabajo con el documento principal y el acceso a sus elementos, pero en cuanto a procesamiento y funcionalidad, como hemos mencionado antes, están listos para la tarea. Por ejemplo, podemos utilizar métodos regulares como setTimeout() o setInterval(), cargar información adicional desde servidores mediante XMLHttpRequest y también acceder a otras API para crear códigos de gran alcance. La última posibilidad es la más prometedora de todas, pero tiene un problema: deberemos aprender una aplicación diferente de las API disponibles para los trabajadores.

Importante

Varias API tienen versiones síncronas, como API File y API IndexedDB, pero la mayoría de ellas se encuentran aún en desarrollo o son inestables. Visite los enlaces en nuestro sitio web para obtener mayor información sobre este tema.

En capítulos anteriores presentamos la implementación asíncrona de algunas API. Sin embargo, la mayoría de las API disponen de versiones asíncronas y síncronas. Estas versiones diferentes de la misma API realizan las mismas tareas pero utilizan métodos específicos según la forma en la que son procesadas. Las API asincrónicas son útiles cuando las operaciones que se realizan requieren mucho tiempo y recursos que el documento principal no puede proporcionar en ese momento. Las operaciones asícronas se realizan en segundo plano mientras que el código principal continúa el procesamiento sin interrupción. Debido a que los trabajadores son nuevos subprocesos que se ejecutan al mismo tiempo que el

código principal, ya son asíncronos y este tipo de operaciones ya no son necesarias.

26.1.6 Importación de scripts

Algo a destacar es la posibilidad de cargar archivos Javascript externos desde un trabajador. Un trabajador puede contener todo el código necesario para realizar cualquier tarea que necesitemos, pero pueden ser creados varios trabajadores para un solo documento, existe la posibilidad de que algunas partes de este código se hagan redundantes. Podemos seleccionar estas partes, ponerlas en un solo archivo y cargar ese archivo desde cada código trabajador con el nuevo método importScripts():

importScripts(archivo): Este método carga un archivo Javascript externo para incorporar el nuevo código en un trabajador. El atributo archivo indica la ruta del archivo que será incluido.

El método importScripts() funciona de forma similar a métodos anteriores de otros lenguajes, como por ejemplo include()de PHP. El código del archivo se incorpora al código como si fuera parte del archivo original.

Para utilizar el nuevo método importScripts() tendrá que declararlo al comienzo del código trabajador, y el código trabajador no estará listo hasta que estos archivos hayan cargado completamente.

```
importScripts('mascodigos.js');

addEventListener('message', recibir);

function recibir(e){
  prueba();
}
```

Código 26-9

Cargar códigos Javascript externos para el trabajador.

El Código 26-9 no es funcional, pero sirve para demostrar cómo usar el método importScripts(). En esta situación hipotética, el archivo mascodigos.js, que contiene la función prueba(), se carga en cuanto se ha cargado el expediente del trabajador. Después de este proceso, la función prueba(), así como cualquier otra función dentro del archivo mascodigos.js, pasa a estar disponible para el resto del código trabajador.

26.1.7 Trabajador compartido

El trabajador que hemos visto hasta ahora se llama trabajador dedicado. Este tipo de trabajador solo responde al código principal desde el cual ha sido creado, pero hay otro tipo de trabajador llamado trabajador compartido, que responde a varios documentos desde el mismo origen. Trabajar con múltiples conexiones significa que podemos compartir el mismo trabajador desde diferentes ventanas, pestañas o marcos, y mantener todo actualizado y sincronizado. La API proporciona un segundo constructor para obtener el objeto SharedWorker:

SharedWorker(scriptURL): Este constructor crea un objeto SharedWorker (trabajador compartido). Sustituye al anterior constructor Worker()utilizado para trabajadores dedicados. Como es habitual, el atributo scriptURL declara la ruta del archivo Javascript que contiene el código para el trabajador. Se puede añadir un segundo atributo opcional para especificar un nombre para el trabajador.

Las conexiones se realizan a través de los puertos, y estos puertos pueden ser guardados en el interior del trabajador para futuras referencias. Para trabajar con trabajadores compartidos y puestos, esta parte de la API incorpora nuevas propiedades, eventos y métodos:

port: Cuando se construye el objeto SharedWorker, se crea un un puerto nuevo para este documento y se asigna a la propiedad port. Esta propiedad será utilizada más adelante para hacer referencia al puerto y comunicarse con el trabajador.

connect: Se trata de un evento específico para detectar nuevas conexiones desde el interior del trabajador. El evento se dispara cada vez que un documento inicia la conexión con el trabajador. Es útil llevar un registro de todas las conexiones disponibles para el trabajador donde se haga referencia a todos los documentos que esté utilizando este trabajador.

start(): Este método está disponible para el objeto MessagePort, que es uno de los devueltos en la construcción de un trabajador compartido, y su función es la de iniciar el envío de mensajes recibidos a través de un puerto. Después de la construcción del objeto SharedWorker, este método debe ser llamado para iniciar la conexión.

El constructor SharedWorker() devuelve un objeto SharedWorker y un objeto MessagePort con el valor del puerto a través del cual se realizará la conexión con el trabajador. La comunicación con el trabajador compartido se debe hacer a través del puerto al que hace referencia la propiedad port.

Para experimentar con Trabajadores compartidos tendremos que utilizar al menos dos documentos diferentes desde el mismo origen: dos códigos Javascript para cada documento y un archivo para el trabajador.

El documento HTML de nuestro ejemplo incluye un iframe que nos permitirá cargar un segundo documento en la misma ventana. Tanto el documento principal como el que mostraremos en el iframe compartirán el mismo trabajador.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Web Workers</title>
  <link rel="stylesheet" href="webworkers.css">
  <script src="webworkers.js"></script>
</head>
<body>
  <section id="formulariocaja">
    <form name="form">
      <label for="name">Nombre: </label>
      <input type="text" name="nombre" id="nombre">
      <input type="button" id="boton" value="Enviar">
    </form>
  </section>
  <section id="cajadatos">
    <iframe id="iframe" src="iframe.html" width="500" height="350"></
 iframe>
  </section>
</body>
</html>
```

Código 26-10

Documento HTML para probar trabajadores compartidos.

El documento que tenemos que crear para el iframe debe ser un documento HTML sencillo con un elemento <section> para recrear nuestra ya tradicional cajadatos y una etiqueta <script> para incluir el archivo iframe.js que contiene el código que se conectará con el trabajador.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>iframe Window</title>
  <script src="iframe.js"></script>
</head>
<body>
  <section id="cajadatos"></section>
</body>
</html>
```

Código 26-11

Documento HTML para el iframe (iframe.html)

Cada documento HTML tiene su propio código Javascript para iniciar la conexión con el trabajador y procesar sus respuestas. Estos códigos tienen que construir el objeto SharedWorker y utilizar el puerto al que hace referencia el valor de la propiedad port para enviar y recibir mensajes. En primer lugar, veamos el código que corresponde al documento principal:

```
var worker;
var worker;
function iniciar(){
  var boton = document.getElementById('boton');
  boton.addEventListener('click', enviar);
  worker = new SharedWorker('worker.js');
  worker.port.addEventListener('message', recibir);
  worker.port.start();
}
function recibir(e){
  alert(e.data);
}
function enviar(){
  var nombre = document.getElementById('nombre').value;
  worker.port.postMessage(nombre);
}
addEventListener('load', iniciar);
```

Código 26-12

Conexión con el trabajador desde el documento principal (webworkers.js).

Si usted tiene un documento en el que quiere trabajar con un trabajador compartido tendrá que crear el objeto SharedWorker y establecer una conexión con el trabajador. En el Código 26-12, el objeto es construido utilizando el archivo worker.js y, a continuación, se establece la comunicación a través del puerto correspondiente utilizando la propiedad port.

Después de añadir un detector para el evento message para detectar las respuestas de los trabajadores, se llama al método start()para comenzar el envío de mensajes. La conexión a un trabajador compartido no se establece realmente hasta que no se ejecute este método. Observe que la función enviar() es similar a la de ejemplos anteriores, excepto que esta vez la comunicación se hace a través del valor de la propiedad port.

Para el iframe, el código no cambia demasiado:

```
function iniciar(){
  var worker = new SharedWorker('worker.js');
  worker.port.addEventListener('message', recibir);
  worker.port.start();
}
function recibir(e){
  var cajadatos = document.getElementById('cajadatos');
  cajadatos.innerHTML = e.data;
}
addEventListener('load', iniciar);
```

Código 26-13

Conexión con el trabajador del iframe (iframe.js).

En ambos códigos, el objeto SharedWorker se construye haciendo referencia al mismo archivo (worker.js), y la conexión se debe establecer usando el la propiedad port (aunque a través de puertos diferentes). La única diferencia notable entre el código del documento principal y el código del iframe es la forma en la que se procesa la respuesta del trabajador. En el documento principal, la función recibir() muestra un mensaje de alerta (ver Código 26-12), mientras que en el iframe, la respuesta se imprime como texto simple dentro de cajadatos (ver Código 26-13).

Ahora es el momento de ver cómo el trabajador compartido maneja cada conexión y envía mensajes al documento apropiado. Recuerde que solo hay un trabajador para ambos documentos (por lo tanto es un trabajador comunitario). Cada solicitud de conexión con el trabajador tiene que ser diferenciada y almacenada para referencias futuras. Vamos a almacenar las referencias a los puertos para cada documento en una matriz llamada puertos.

```
var puertos = new Array();
addEventListener('connect', conectar);


function conectar(e){
  puertos.push(e.ports[0]);
  e.ports[0].onmessage = enviar;
}
function enviar(e){
  for(var f = 0; f < puertos.length; f++){
    puertos[f].postMessage('Tu nombre es ' + e.data);
  }
}
```

Código 26-14

El código Javascript para el trabajador común (worker.js).

Este procedimiento es similar al de los trabajadores dedicados. Esta vez tenemos que considerar a qué documento que vamos a responder, porque varias de ellas pueden ser conectadas al trabajador al mismo tiempo. Para ello, el evento conectar proporciona la matriz puertos con el valor del puerto recién creado (la matriz solo contiene este valor situado en el índice 0).

Cada vez que un código solicite una conexión con el trabajador, se disparará el evento conectar. En el Código 26-14, este evento llama a la función conectar(). En esta función se realizan dos operaciones: en primer lugar, el valor del puerto es tomado de la propiedad ports (en el índice 0) y luego es almacenado en la matriz ports (inicializada al comienzo del trabajador). En segundo lugar, la propiedad del evento onmessage está registrada para este puerto en particular, y la función enviar() está lista para a ser llamada cuando se reciba un mensaje.

Como resultado, cada vez que se envía un mensaje al trabajador desde el código principal, independientemente de cuál sea el documento de origen, se ejecuta la función enviar() del trabajador. En esta función, se utiliza un bucle for para devolver todos los puertos abiertos para este trabajador a partir de la matriz puertos y enviar un mensaje a cada documento conectado. El proceso es el mismo para los trabajadores dedicados, pero esta vez se responde a varios documentos en lugar de hacerlo solo a uno.

Hágalo usted mismo

Para probar este ejemplo, deberá crear varios archivos y subirlos a su servidor. Cree un archivo HTML con el Código 26-10. Este documento cargará el mismo archivo webworkers.css que hemos venido usando a lo largo de este capítulo, el archivo webworkers.js con el Código 26-12 y el archivo iframe.html como la fuente del iframe, con el Código 26-11. También tiene que crear un archivo llamado worker.js para el trabajador con el Código 26-14. Una vez que todos estos archivos hayan sido guardados y cargados al servidor, abra el primer archivo en su navegador. Utilice el formulario para enviar un mensaje al trabajador y vea cómo ambos documentos (el documento principal y el documento del iframe) procesan la respuesta.

Conclusiones

Trabajando para el mundo

Este libro es sobre HTML5. Es una guía para desarrolladores, diseñadores y programadores que deseen crear sitios web y aplicaciones revolucionarias; para el genio interior que todos tenemos oculto; para los autores intelectuales. Pero estamos en una transición, un momento en el que las viejas tecnologías se están fusionando con otras nuevas, y los mercados se están quedando atrás. Al mismo tienpo que millones y millones de copias de nuevos navegadores se descargan desde la Web, millones y millones de personas no son conscientes de su existencia. El mercado está lleno de viejos equipos que utilizan Windows 98 e Internet Explorer 6, y cosas aún peores.

La creación de contenidos para la Web ha sido siempre un reto. A pesar de los grandes esfuerzos por construir e implementar estándares para la Web, ni siquiera los nuevos navegadores los soportan siempre. Y los navegadores antiguos, que no han implementado los estándares, siguen siendo utilizados en todo el mundo y dificultando el trabajo de los desarrolladores.

Así que es el momento de discutir qué podemos hacer para acercar HTML5 al público, para innovar y crear, y para ser verdaderos genios en un mundo que nos trata con indiferencia. Es el momento de ver cómo podemos trabajar con las nuevas tecnologías y ponerlas a disposición de todos.

Las alternativas

Cuando nos encontramos con alternativas, tenemos que decidir qué camino tomar. Podemos ser trabajadores groseros, educados, inteligentes o dedicados. Un desarrollador grosero diría: “Mire, esta página fue programada para trabajar en nuevos navegadores. Los nuevos navegadores son gratuitos, no sea perezoso y descargue una copia”. El desarrollador educado diría: “Esto ha sido desarrollado aprovechando las nuevas tecnologías. Si quiere disfrutar de mi trabajo con todo su potencial, actualice su navegador. Mientras tanto, aquí está una versión que puede utilizar con navegadores anteriores”. El desarrollador inteligente diría:” Hacemos que la tecnología más avanzada esté disponible para todos. Usted no tiene que hacer nada porque ya lo hemos hecho por usted”. Por último, un desarrollador dedicado diría: “Ésta es una versión de nuestro sitio web adaptada a su navegador. Aquí hay otra versión con más funciones especiales para los nuevos navegadores y ésta es una versión experimental de nuestra evolucionada aplicación qye solo está disponible para la versión beta de este navegador específico.”

Para hacer un enfoque más útil y práctico, digamos que hay cuatro opciones disponibles cuando el navegador del usuario no está preparado para HTML5:

Informar: Pida al usuario que actualice el navegador si no están disponibles algunas de las características de que su aplicación necesita.

Adaptar: Seleccione diferentes estilos y códigos para el documento de acuerdo con las características disponibles en el navegador del usuario.

Redireccionar: Dirija el usuario a un documento completamente nuevo diseñado especialmente para navegadores antiguos.

Emular: Utilice las bibliotecas para hacer que las características de HTML5 estén disponibles en navegadores antiguos.

Modernizr

Independientemente de la opción que elija, el primer paso es detectar si las características de HTML5 que requiere su aplicación están disponibles para el navegador del usuario. Las características son independientes y fáciles de identificar, pero las técnicas necesarias para detectarlas son tan diversas como las mismas características. Los desarrolladores deben considerar los diferentes navegadores y sus versiones, así como los códigos que a menudo son poco confiables.

Una pequeña biblioteca llamada Modernizr ha sido desarrollada para resolver este problema. Esta biblioteca crea un objeto llamado Modernizr que ofrece propiedades para todas las características de HTML5. Estas propiedades devuelven un valor booleano que será verdadero o falso en función de la disponibilidad de las características.

La biblioteca es de código abierto, fue realizada en Javascript y está disponible de forma gratuita en www.modernizr.com. Solo hay que descargar el archivo Javascript e incluirlo en el encabezado del documento, como se muestra en el siguiente ejemplo:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Modernizr</title>
  <script src="modernizr.min.js"></script>
  <script src="codigo.js"></script>
</head>
<body>
  <section id="cajadatos">
    contenido
  </section>
</body>
</html>
```

Código C-1

Incluir Modernizr en nuestros documentos.

El archivo llamado modernizr.min.js es una copia del archivo de la biblioteca Modernizr, que ha sido descargado desde el sitio web Modernizr (www.modernizr.com). El segundo archivo incluido en el documento HTML del Código C-1 es nuestro propio código Javascript, creado para comprobar los valores de las propiedades presentadas por la biblioteca.

```
function iniciar(){
  var cajadatos = document.getElementById('cajadatos');
  if(Modernizr.boxshadow){
  cajadatos.innerHTML = 'La sombra de la caja está disponible.;
  }else{
  cajadatos.innerHTML = 'La sombra de la caja no está disponible';
  }
}
addEventListener('load', iniciar);
```

Código C-2

Detectar la disponibilidad de los estilos CSS para las sombras de la caja.

Como se puede ver en el Código C-2, es posible detectar cualquier característica de HTML5 utilizando una declaración if y la propiedad correspondiente del objeto Modernizr. Cada característica tiene una propiedad disponible.

Importante

Ésta es una breve introducción a una útil biblioteca. Mediante el uso de Modernizr, por ejemplo, también es posible seleccionar un conjunto de estilos CSS de determinados archivos CSS sin usar Javascript. Modernizr ofrece también clases especiales para aplicar en hojas de estilo con el fin de seleccionar las propiedades CSS adecuadas de acuerdo con las características disponibles. Para obtener más información, visite www.modernizr.com.

Bibliotecas

Una vez que son detectadas las características disponibles, tiene la opción de utilizar la información detectada para trabajar en el navegador del usuario o recomendarle a éste que actualice su software. Sin embargo, suponga que usted es un desarrollador terco o un programador loco a quien (junto con sus usuarios y clientes) no le importan los proveedores o la versión del navegador, las versiones beta, las características no implementadas ni nada más que ejecutar la última tecnología disponible, sin importar nada más.

Bueno, aquí es donde pueden resultar de ayuda las bibliotecas independientes. Muchos programadores de muchos puntos del planeta son, probablemente, más tercos que usted y sus clientes, y están desarrollando y mejorando las bibliotecas para emular las características de HTML5 en navegadores antiguos, especialmente la API de Javascript. Gracias a sus esfuerzos, hoy contamos con los nuevos elementos HTML, los selectores y estilos CSS3 e incluso algunas API tan complejas como Canvas o Web Storage, disponibles en todos los navegadores del mercado.

Estas bibliotecas se llaman polyfills, y puede encontrar una lista actualizada en la siguiente dirección:

www.github.com/Modernizr/Modernizr/wiki/HTML5-Cross-browser-Polyfills.

Google Chrome Frame

Google Chrome Frame es probablemente el último recurso al q ue deberá recurrir. Inicialmente fue una buena idea, pero ahora es mejor recomendar a los usuarios que actualicen sus navegadores que hacerlos descargar un plug-in.

Google Chrome Frame es un plug-in que ha sido desarrollado específicamente para versiones anteriores de Internet Explorer. Fue diseñado para proporcionar todo el poder y las posibilidades de que Google Chrome a los navegadores que no están preparados para estas tecnologías. Mediante la inserción de una sencilla etiqueta HTML en los documentos, se muestra a los usuarios un mensaje que recomienda la instalación de Google Chrome Frame antes de ejecutar su sitio web o aplicación. Después de este simple paso, todas las funciones compatibles con Google Chrome pasan a estar disponibles automáticamente. Sin embargo, en vista de que los usuarios tienen que descargar el software desde la Web, ¿por qué no recomendarles descargar una nueva versión del navegador directamente? Sobre todo ahora que Internet Explorer tiene su propia versión compatible con HTML5, que es disponible de forma gratuita. Ahora que hay tantos navegadores que soportan HTML5, es mejor guiar a los usuarios a estos nuevos software en lugar de enviarlos a confusos plug-ins.

De cualquier manera, aunque ésta no debería ser su primera alternativa, podría serle útil en algunas circunstancias. Para saber más acerca de Google Chrome Frame y aprender a utilizarlo, visite http://code.google.com/chrome/chromeframe/.

Trabajar para la nube

En este nuevo mundo de dispositivos móviles y computación en la nube, independientemente de la actualidad del navegador, siempre tendremos algo nuevo de qué preocuparnos. Probablemente todo comenzó con el iPhone: desde su lanzamiento han cambiado varias cosas para la Web. Más adelante han surgido todo tipo de dispositivos, como el iPad entre otros, para expandir este nuevo mercado. Gracias a este cambio radical en el mundo electrónico, el acceso a Internet móvil se ha convertido en algo común. Estos nuevos dispositivos son ahora un foco importante para los sitios y aplicaciones web, y su diversidad de plataformas, pantallas e interfaces ha obligado a los desarrolladores a adaptar sus productos a cada caso concreto.

En estos días, independientemente de qué tipo de tecnología que utilcemos, los sitios y aplicaciones web que desarrollemos deben adaptarse a todas las plataformas posibles para mantener la coherencia y lograr que nuestro trabajo esté a disposición de todos. Afortunadamente, HTML siempre ha considerado esta situación y proporciona el atributo media en el elemento <link> para seleccionar recursos externos de acuerdo con predeterminados parámetros:

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Documento principal</title>
  <link rel="stylesheet" href="webstyles.css" media="all and (min-
                    width:769px)">
  <link rel="stylesheet" href="tablet.css" media="all and (min-
                    width:321px) and (max-width: 768px)">
  <link rel="stylesheet" href="phone.css" media="all and (min-width:
                          0px)and (max-width: 320px)">


</head>
<body>


  "
</body>
</html>
```

Código de C-3

Incluir diferentes archivos CSS para distintos dispositivos.

Seleccionar distintos estilos CSS es una manera fácil de lograr nuestro objetivo. Los archivos CSS se cargan y se aplican los estilos apropiados según el dispositivo o el tamaño de la pantalla. Los elementos HTML se pueden redimensionar, de manera que documentos enteros pueden ser adaptados y representados de acuerdo a un espacio y unas circunstancias específicas.

En el Código C-3 se incorporan tres archivos CSS diferentes para tres situaciones diferentes. Las situaciones son detectadas por los valores del atributo media en todas las etiquetas <link>. Mediante el uso de las propiedades min-width y max-width podemos determinar el archivo CSS que se aplicará a cada documento de acuerdo a la resolución de la pantalla en la que se muestra el documento. En este ejemplo se consideran tres dispositivos: un smartphone pequeño, un tablet PC y un ordenador de tamaño estándar. Los valores utilizados se encuentran generalmente en estos dispositivos.

Si el tamaño horizontal de la pantalla está entre 0 y 320 píxeles, se carga el archivo phone.css para representar el documento; para una resolución que esté entre 321 y 768 píxeles, se carga el archivo tablet.css y, por último, para una resolución de 768 píxeles o mayor se carga el archivo webstyles.css.

Por supuesto, el proceso de adaptación no implica solo los estilos CSS. Las interfaces que proporcionan estos dispositivos son ligeramente diferentes a las de un ordenador de mesa, porque algunas partes físicas, como pueden ser las palabras clave o el ratón, se eliminan. Eventos habituales como clic o mouseover

han sido modificados o, en algunos casos, reemplazados por eventos táctiles. Y hay otra importante característica, normalmente presente en los dispositivos móviles, que permite al usuario cambiar la orientación de la pantalla y, por lo tanto, el espacio disponible para el documento.

Todos estos cambios respecto a la interfaz de un ordenador tradicional hacen que sea casi imposible conseguir una buena adaptación del diseño y operatividad de un sitio o aplicación web con solo añadir o modificar algunas reglas CSS. Es necesario utilizar Javascript para adaptar los códigos o, incluso, detectar la situación y redirigir a los usuarios a una versión de la página web creada específicamente para el dispositivo que accede a la aplicación.

Importante

El tema aquí tratado va más allá del alcance de este libro. Para obtener más información, por favor visite nuestro sitio web y siga los enlaces de este capítulo.

Las API que no han sido incluidas

Por diferentes razones, no ha sido posible incluir todas las API proporcionadas por HTML5 en este libro. Pero algunas de ellas son lo suficientemente importantes como para que por lo menos sepas que están disponibles:

API Clipboard: Esta API proporciona acceso al portapapeles del sistema. Ofrece eventos para controlar cada acción posible (copiar, cortar y pegar) y también acceso a una interfaz utilizada en la API Drag and Drop para establecer y recuperar los datos involucrados en el proceso.

API Device Orientation: Esta API proporciona acceso a la información ofrecida por el dispositivo sobre la orientación y el movimiento.

API Quota Management: Como mencionamos en el Capítulo 16, la API Quota Management permite a las aplicaciones solicitar espacio de almacenamiento o comprobar el espacio disponible.

API SVG: Esta API, cuyo nombre viene de Scalable Vector Graphics (Vectores gráficos escalables) brinda la posibilidad de generar gráficos vectoriales para la web. Debido al éxito de Flash (un plug-in de gráficos vectoriales), esta API se veía en un principio como una buena opción, pero hay un inconveniente. A diferencia del resto de las especificaciones, ésta se basa en XML en lugar de Javascript. Mientras todo el mundo intenta simplificar el marcado, esta API ha introducido una mayor complejidad al seguir por un camino totalmente diferente. Es por esta razón que no ha sido incluida en este libro, pero usted debe evalua¡la por sí mismo.

HTML5 es considerado casi un estándar de vida. Sus especificaciones están en constante cambio y expansión. En la sección de enlaces de la página web de MinkBooks, encontrará enlaces a todas las páginas de la especificación. Visite

estas páginas con frecuencia para comprobar el estado actual de cada una y para ampliar sus conocimientos sobre estas tecnologías.

Lo que debe saber

Las tecnologías que se explican en este libro, especialmente el lenguaje Javascript, son extremadamente flexibles y pueden aplicarse de tantas maneras que es imposible para mostrarle todas las opciones disponibles. Sin embargo, hay algunas cosas que debe saber:

La manera en la que organizamos nuestro código JavaScript y cómo aplicamos las herramientas proporcionadas por el lenguaje suelen ser llamadas patrones (patterns). En este libro hemos implementado un patrón muy simple, pero existen innumerables alternativas disponibles, e incluso libros enteros que se centran en este tema. Deberá aprender más acerca de los patrones para decidir cuál es el mejor para sus proyectos.

La propiedad prototype, proporcionada por Javascript para modificar los prototipos de una cadena de prototipos, es ampliamente utilizado en la Web. El modelo de programación aplicado en nuestros ejemplos evita el uso de esta propiedad, pero es importante que aprenda más sobre ella.

En este libro solo hemos hecho tratado de forma muy superficial algunas API muy complejas. Para describir todas las características de algunas de ellas, sería necesario un libro entero. Le recomendamos que cuando tenga que ponerlas en práctica, amplíe antes sus conocimientos mediante la revisión de las especificaciones oficiales (visite la sección de enlaces de nuestro sitio web, www.minkbooks.com).

Palabras finales del autor

Siempre habrá desarrolladores que digan: “Si utiliza tecnologías que no están disponibles para el 5% de los navegadores del mercado, perderá el 5% de los clientes”. Mi respuesta es: “usted tiene clientes que satisfacer, entonces adapte, reoriente o emule, pero si trabaja usted mismo, solo informe”.

Siempre hay que encontrar la manera de tener éxito. Si trabaja para los demás, para tener éxito deberá ofrecer una solución completa, un producto al que pùedan acceder los clientes de su cliente sin importar lo que los ordenadores, navegadores o sistemas que utilizan. Pero si usted trabaja para sí mismo, tendrá que crear el mejor producto; deberá innovar y estar a la cabeza de todos, independientemente de lo que el 5% de los usuarios hayan instalado en sus equipos. Hay que trabajar para el 20% que ya han descargado la última versión de Mozilla Firefox, el 15% que ya ha ejecuta Google Chrome en su PC y el 10% que tiene Safari en sus dispositivos móviles. Hay millones de usuarios dispuestos a convertirse en sus clientes. Mientras que el desarrollador le pregunta por qué pierde el 5% del mercado, yo le pregunto por qué tendría que perder la oportunidad de triunfar.

Nunca capturará el 100% del mercado, y eso es un hecho. No está desarrollando sitios web en chino o portugués y no está trabajando para el 100% del mercado sino para una pequeña parte de éste. ¿Por qué seguir limitandose a sí mismo? Desarrolle para el mercado que le permite tener éxito. Desarrolle para la parte del mercado que está creciendo continuamente y que le permitirá liberar su genio interior. De la misma manera que no se preocupa por los mercados que hablan otros idiomas, no se preocupe por la parte del mercado que sigue utilizando las viejas tecnologías. Infórmelos. Hágales saber lo que se están perdiendo. Tome ventaja de las últimas tecnologías disponibles y sea un genio. Desarrolle para el futuro y alcanzará el éxito.